

JavaTM magazine

By and for the Java community 

THE INTERNET OF THINGS JAVA IS EVERYWHERE

13

HENRIK STÅHL
ON JAVA AND IOT

29

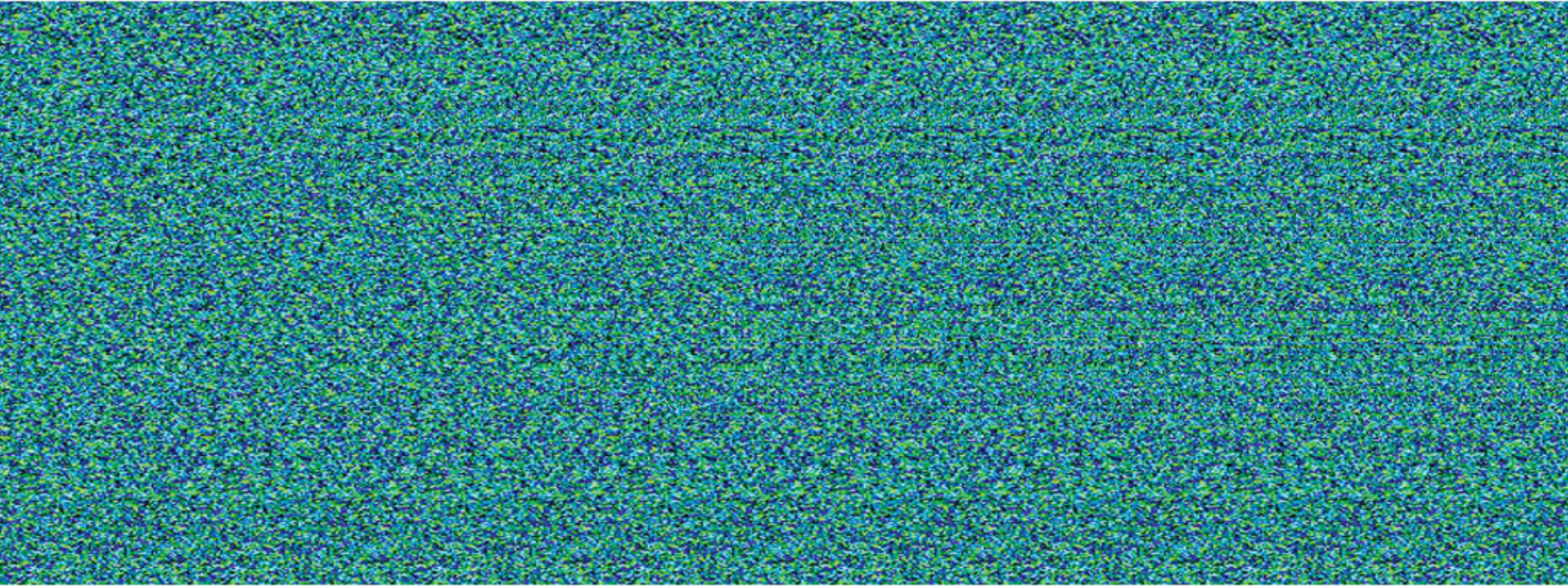
INDUSTRIAL AUTOMATION
WITH ROBOTS

+

20

JAVA: THE NEXT
GENERATION

The answer is right in front of you



Java Image Enabling SDKs that Help You See the Big Picture

At first glance it may seem difficult, but it's really quite simple. Atalasoft's JoltImage product is a proven SDK for image enabling your Java-based web applications, easily. Image enabling helps to add dimension to your data, so you can uncover insights such as correlations and causations hidden inside your 2-dimensional documents. Our SDK does the heavy lifting for you, saving time, money, and the headaches of figuring it out yourself. Backed by our highly knowledgeable & caffeinated support engineers, JoltImage will enable your success and make the big picture so much easier to see.



[Click for tips on viewing the stereogram](#)

Image enabling experts & bacon connoisseurs. Visit us online to see our full line of SDK products for .NET and Java



JAVA IN ACTION



ABOUT US

○

f

ava
net

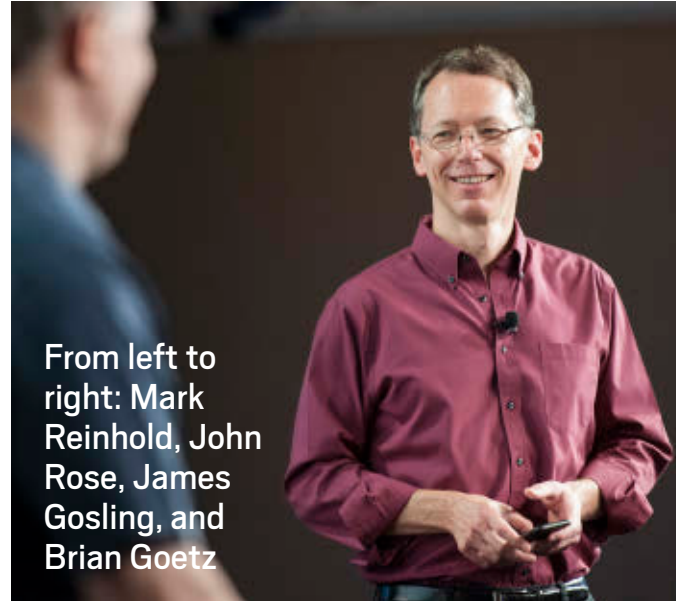
log



Watch the Strategy keynote.

Java skills to use those skills in the embedded space and the growing IoT.

Later, Oracle's **Cameron Purdy** emphasized how the Java platform is continuously evolving toward end-to-end capabilities. He acknowledged the help of the community and emphasized that Java EE 7, the only standards-based platform for enterprise development, is continuing to drive down the costs of building and maintaining applications. He pointed out that the JCP Executive Committee has unanimously approved the roadmap to Java EE 8—which was drawn up directly from the survey responses of 4,500 Java developers. Purdy observed that for Java EE 7, 19 JUGs made valuable contributions by adopting 14 JSRs, and he called for that kind of community participation going forward.



From left to right: Mark Reinhold, John Rose, James Gosling, and Brian Goetz



TECHNICAL KEYNOTE

The Technical keynote was held in two parts: one that followed the Strategy keynote and a second that preceded the Community keynote. Oracle's **Mark Reinhold** reflected on Java as it nears its 20th birthday about ways it can keep going for another 20 years. He addressed the challenges of enhancing Java's ability to scale down, improve security, evolve more easily, improve startup performance, and scale up to large systems. To address these issues, Java 9 is focused on Project Jigsaw, which is "an attempt to design and implement a standard module system for the Java SE platform and use that to modularize the platform itself—and applications," he said. Jigsaw will have a profound impact on how large Java systems are built and maintained and improve developer performance and productivity, Reinhold said.

Next, **Brian Goetz** offered a vision of Java extending to Java 9 and beyond that would

include value classes. Look to Project Valhalla and Project Panama for more information, he said.

In part two, Reinhold gathered a panel of Java luminaries, including **James Gosling**, onstage to take questions. When asked when Java would become obsolete, Gosling confessed that for a decade he has been expecting Java's demise, but that Java is a kind of organism grounded in the community that is well understood and flexible and has strong staying power.



Watch the Technical keynote.



Bruno
Maisonnier



Paul Perrone



Andra Kay

COMMUNITY KEYNOTE

At the JavaOne Community keynote, Java Evangelist James Weaver gathered a large group of innovators, who showed off Java-based projects—everything from programs that teach kids to code to automated vehicle testing systems that can save lives.

Robotics was a key theme. Some highlights: **Andra Kay**, director at Silicon Valley Robotics, said, "By 2020 your household robot will be your house."

Bruno Maisonnier, CEO at Aldebaran, a world leader in humanoid robots, presented a video showing robots teaching children mathematics in schools.

Paul Perrone of Perrone Robotics lamented the 30,000 deaths from auto accidents each year

in the United States, and showed a video about his automated vehicle testing system with an advanced braking system that could save lives—a first step toward cars with full autonomy.

The Community keynote ended with the traditional T-shirt toss (or catapult), led by **James Gosling**. It was the perfect ending to a great week of information sharing, learning, and community building.

Watch the [Community keynote](#).

NullPointerException Rock JavaOne



The NullPointers, the unofficial Java community band, rocked JavaOne at the JCP party and later in the week at Duke's Café. The band is made up entirely of JavaOne community members.



FEATURED JAVA USER GROUP

UTAH JAVA USERS GROUP

The Utah Java Users Group (UJUG) was founded in 1998 by **Ben Galbraith** and **Glen Lewis**. The group is currently led by **Jason Porter** and supported by board members **Don Bogardus**, **Mike Bylund**, and **Chris Hansen**. UJUG currently has about 1,000 members. Meetings, which are normally held on the third Thursday of each month, typically draw from 80 to 180 attendees.

If this level of participation surprises you (because Utah is in the bottom 20 percent of the United States in population density), that probably means you haven't heard of "Silicon Slopes" ([@SiliconSlopes](#)), the cluster of infor-

mation technology and software development firms along the Wasatch Front in north central Utah that is "leading the world in innovative memory process technology. . .and data analysis," according to [CNBC](#).

Porter notes that, indeed, the Silicon Slopes phenomenon has had an enormously positive effect on UJUG's growth and vivacity. "Silicon Slopes has seen a tech boom, and Java is the language of choice in all the major shops, and most of the smaller ones," he says. "This makes for great local presenters, and we also get visits from out-of-towners, including **Jason Van Zyl**, **Gavin King**, and **Venkat**

Subramaniam."

Each UJUG meeting has two presentations, and the presenter schedule is already filled into late 2015.

Meetings typically start with the traditional pizza, and they often end with a trip to the local pub for those interested.

"Yes, there's good beer in Utah!" Porter exclaims. He adds that each UJUG meeting is unique. "Something new is tried each month to improve networking and for just plain fun—we had a paper airplane competition last month."

If you're a Java developer who loves skiing or the great outdoors, you may want to consider the Silicon Slopes next time you're seeking new work. "There's a healthy startup community and a trend for big out-of-state shops to open local development centers," says Porter. "Developer unemployment in Utah is down to 1.5 percent, with salaries continuing to rise (the latest trend is highway recruiting billboards). All the energy in the ecosystem makes for great UJUG meetings, and a tangible excitement in the community."

Learn more at UJUG's [Meetup site](#).

JAVA DEVELOPMENT FOR THE INTERNET OF THINGS

Oracle's **Henrik Ståhl**
discusses the Internet of
Things for Java developers.
BY TIMOTHY BENEKE



The Internet of Things (IoT) is poised to blow into our daily lives and make many of the things we interact with more efficient, faster, and more fun. Analysts predict that by 2020 there will be 50 billion devices wirelessly connected to each other and making smart decisions for us. We're seeing early adoption in healthcare, including lifestyle health devices, patient monitoring, and home healthcare. Another big area is telematics in the automotive industry—not to mention home automation. Suffice it to say that everything from our homes to healthcare to the workplace will change due to the IoT.

For Java developers, many opportunities and questions present themselves. How can they deploy their present skills to better apply them to IoT development? What changes

We met up with Henrik Ståhl, vice president of product management for Java and the IoT at Oracle, to discuss these and other issues.

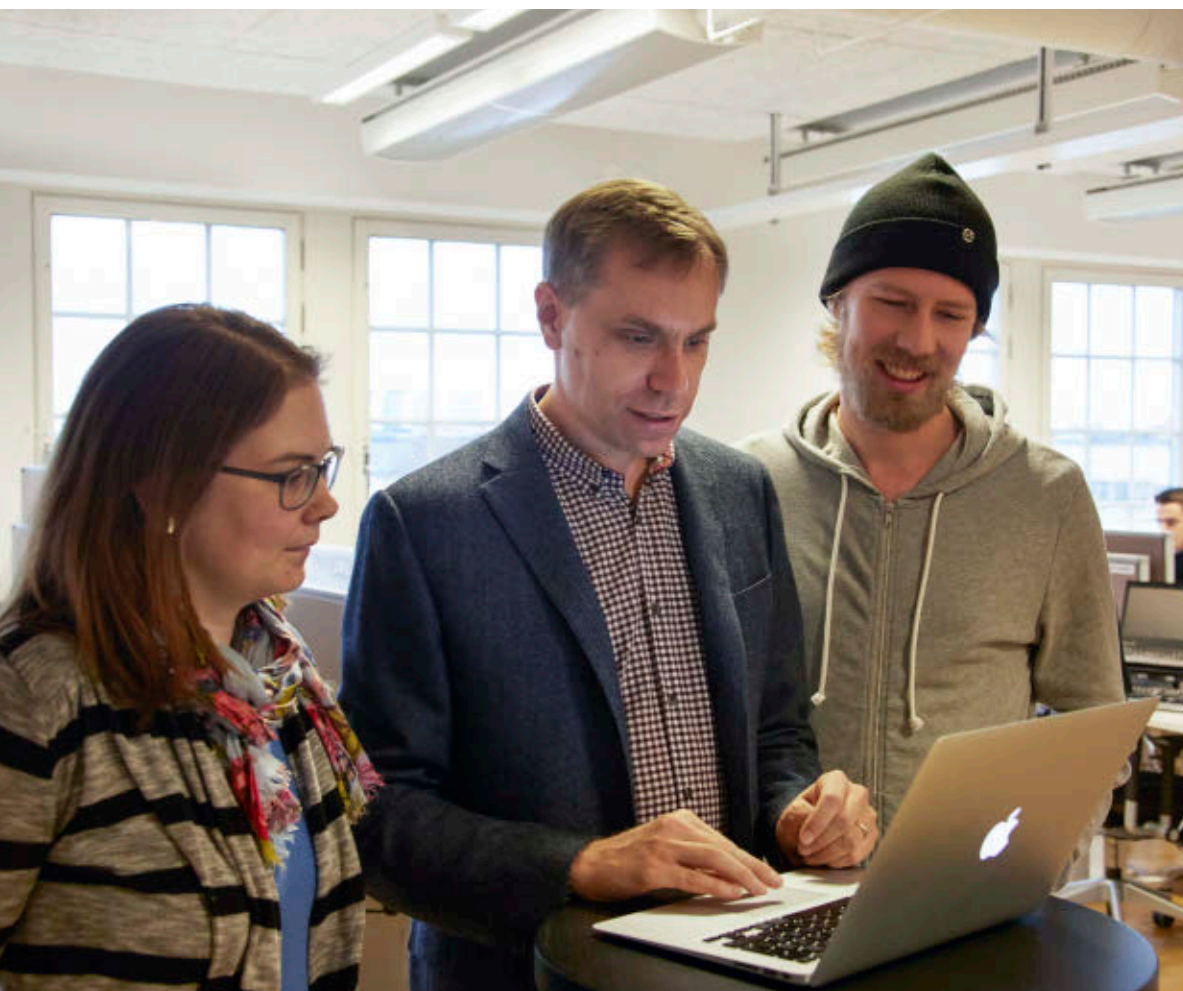
Stahl: Java developers can already use their skills to develop back-end applications for the IoT. We want them to also be able to develop software on the various devices that constitute the devices—the *things* in the IoT—using the same language, the same APIs, and compact runtimes suitable for the embedded space, which in the past has involved a steep learning curve for everyone who ventures into it. Instead, developers have had to learn low-level languages that impede productivity and lead to relatively high maintenance costs. Now, because embedded development often includes integration with various types of specialized peripherals and sensors, it will likely always require some amount of low-level programming. We are not aiming to replace that, but rather to enable an application developer to work in Java, freeing up the system developer to focus only on the hardware enablement. This distinction

To achieve this vision, it is vital that we get to a common programming model across all devices, regardless of size or device capabilities. With the release of Oracle Java ME Embedded 8 and Oracle Java SE Embedded 8, we have made the features that developers are used to in Java available on embedded platforms with as little as a couple of hundred kilobytes of memory. A major goal of these two releases was

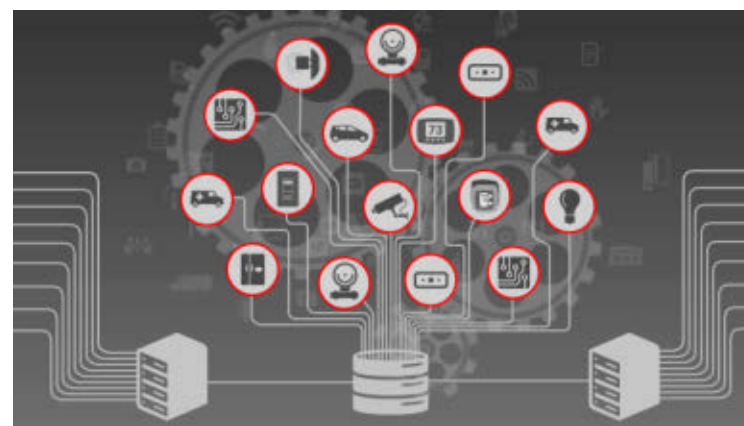
To get toward this goal while achieving low memory and disk footprints, we have introduced the concept of Compact Profiles in Java SE 8. These give developers a choice of deploying the entire set of Java SE APIs, or one of three different subsets that are incrementally smaller. These subsets have been selected very carefully to match what APIs are needed for common use

Henrik Ståhl at work
at Oracle's Stockholm,
Sweden, office





cases such as running an OSGi [Open Service Gateway initiative] stack. The Compact Profiles are a stepping-stone toward more-granular modularity that will be introduced with Project Jigsaw in Java SE 9.



Watch the video: *Java Embedded for IoT*

These embedded Java platforms are the number one thing that we are providing to embedded developers.

And as we include tools—for example, on the Java ME side so that it catches up with the likes of Java SE—it will be really easy for the 9 million Java developers to start taking on embedded projects, which is something that we are obviously trying to encourage. We have seen very good uptake of Java already in the higher-end devices, such as wireless modules (small chips that integrate a low-power CPU, some memory, and a 3/4G modem for communications) and larger SoCs [systems on a chip] including the popular hobbyist-targeted

Raspberry Pi. Our traction in the micro-controller space is more limited to date, but with the release of Oracle Java ME Embedded 8 we believe that our platform is well suited to meet market demands. And so, we think that with enough energy and support coming from the Java community and Oracle, we can get Java developers to start focusing on embedded. This brings an immediate benefit to the IoT because Java greatly facilitates key device-side requirements such as communications, integration, security, and a flexible application model. The notion of having billions of things out there on constrained form factors means that we need programming models to push business logic to the network edge and make decisions at the perimeter of the network as opposed to centralizing all communication behind the customer firewall and the cloud. So, the embedded Java platform is only going to

“Java developers can already use their skills to develop back-end applications for the IoT. **We want them to also be able to develop software on the various devices that constitute the devices in the IoT.**”

Stahl: With our embedded platforms, Oracle is enabling all of the applications in the Oracle portfolio and all of the applications being built by the Java community to talk to the IoT. So if you

I am quick to caution that it's not necessarily the biggest value proposi-



So embedded Java becomes the enabling technology. For developers who pay attention, this will allow them to tap into an entirely new class of appli-

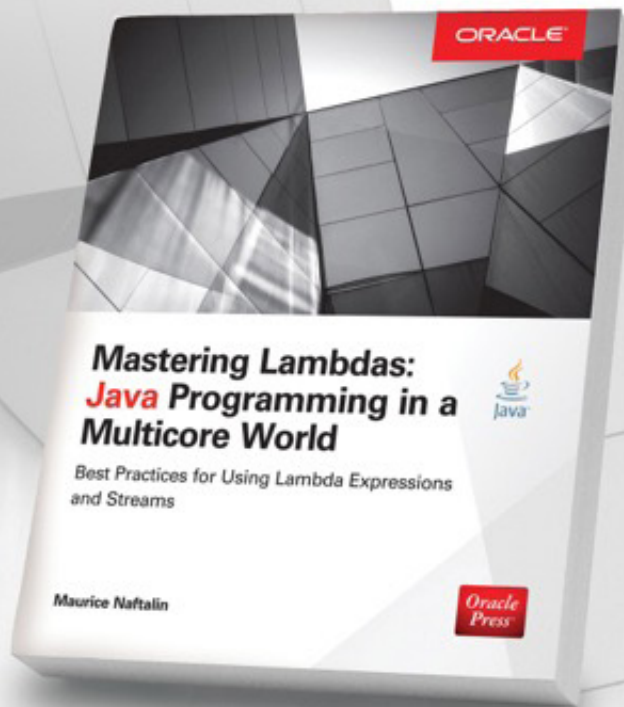
Written by leading Java experts, Oracle Press books offer the most definitive, complete, and up-to-date coverage of Java available.



Iron-Clad Java: Building Secure Web Applications

Jim Manico, August Detlefsen

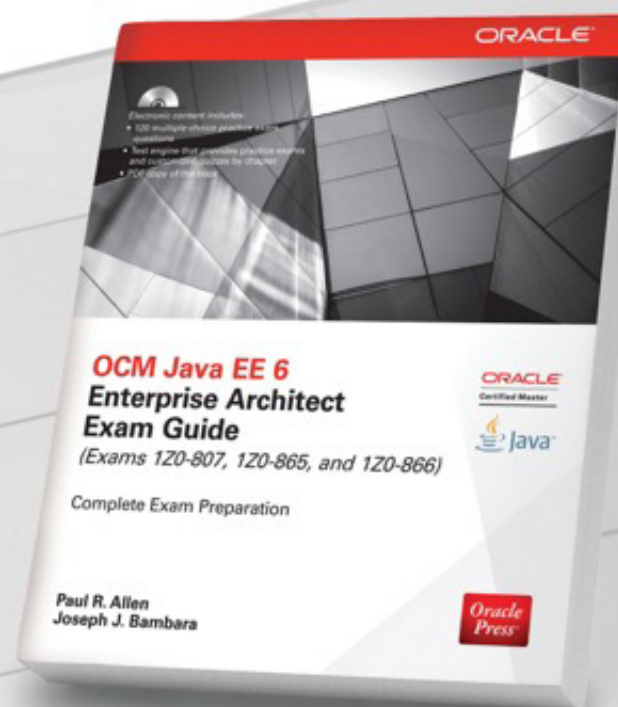
Develop, deploy, and maintain secure Java applications using the expert techniques and open source libraries in this Oracle Press guide.



Mastering Lambdas: Java Programming in a Multicore World

Maurice Naftalin

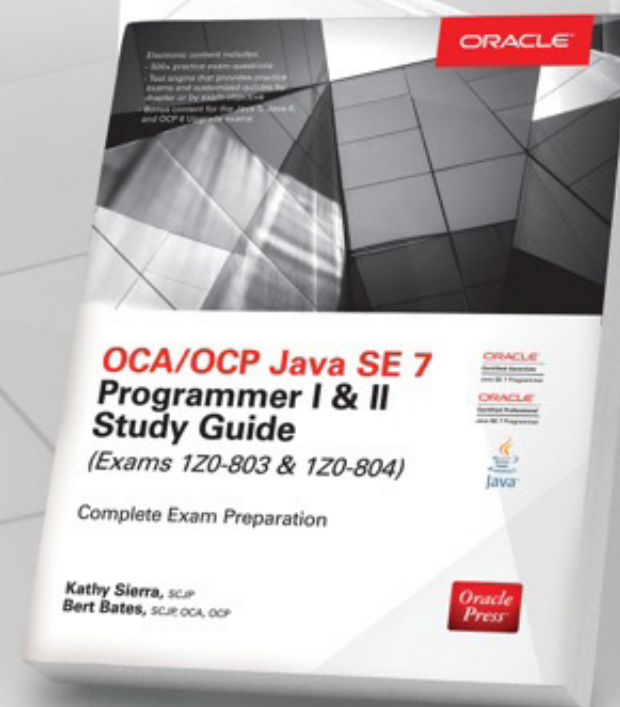
Effectively use lambda expressions to maximize performance improvements provided by multicore hardware.



OCM Java EE 6 Enterprise Architect Exam Guide

Paul Allen, Joseph Bambara

Get complete details on Oracle Certified Master Java EE 6 Enterprise Architect Exams 1Z0-807, 1Z0-865, and 1Z0-866. 120 multiple-choice practice exam questions are included.



OCA/OCJP Java SE 7 Programmer I & II Study Guide

Kathy Sierra, Bert Bates

This self-study tool offers full coverage of OCA/OCJP Exams 1Z0-803 and 1Z0-804. Electronic content includes 500 practice exam questions.



 Kids at the Create the Future Java Workshop talk with Caroline Kvitka.

science students in the United States alone by 2020.

"Learning to program actually is learning to decompose a problem into simpler steps," Dann says. "And by accomplishing those steps, one

That's where extracurricular programs come in. Although opportunities for kids to learn to code were limited just a few years ago, today programs abound and kids (and sometimes their parents) are flocking to them.

Alice 2.0 focuses on logical and computational thinking skills and programming fundamentals, while Alice 3.0 emphasizes object-oriented concepts and a full transition to the Java programming language. LEGO Mindstorms combines building LEGO robots with programming.

"We have used Java as our tool for implementation, and we write our code so that after students have used Alice they can go into a classroom



Left: Kids took a stretching break at the Devovx4Kids event. Below: Chris Hollinger of Oracle Academy talked about Alice at the Create the Future Java Workshop.



Devovx4Kids can help him get where he needs to go, he adds.

Of the Devovx4Kids attendees, 21 were girls from the San Francisco chapter of [Black Girls Code](#), an organization that seeks to introduce girls from underserved communities to computer science and technology with a goal to increase their representation in the STEM (science, technology, engineering, and mathematics) fields. "The event was a wonderful opportunity for our students to experience a wide array of technology tools in a youth-friendly


atmosphere," says Black Girls Code Founder Kimberly Bryant.

AROUND THE WORLD

Computer programs for kids are popping up all over the world. In the Philippines, [JEDI4KiDS](#) educates children to be more creative using computers and provides them the opportunity to learn computer programming in a fun and interactive manner.

JEDI4KiDS is part of the larger JEDI initiative, which provides free and open source computer science and IT



 Kids and organizers talk about their experience at the Devovx4Kids event.



Maulin Patel with team members Greg Hemstreet and Mary Thomas at Freescale's Austin, Texas, headquarters.



specifically the interplay between Java and the IoT framework. The entire Java community needs to become more aware of IoT issues, because growth is going to be substantial. And the whole Java ecosystem will be strengthened by adding additional effort and attention to these areas.

Patel: The Java architecture allows any customer to create vertical applications that span data collection at the edge nodes, to the gateway communication layer, and up to the cloud, using a

“Security issues are of paramount importance at this stage of the IoT’s development, specifically the interplay between Java and the IoT framework.”

consistent architecture that all plays together seamlessly. That's a huge advantage. You can begin data collection with Java ME, move to Java SE for gateway services, and move to Java EE for data management and processing. This frees Java IoT developers from worrying about hardware dependencies and allows them to create applications much more quickly.

Java Magazine: Are there any changes you would like to see in the JCP?

As a functional body, the JCP is working well. But we need to make participation easier for individuals who may not be affiliated with large corporations. Many of these developers have brilliant IoT and Java-related ideas and expertise to contribute. Also, Java user groups span the globe. We need to incentivize this amazing pool of talent to bring more JSRs to the table.

Java Magazine: What are your thoughts about attracting existing Java developers who are not very familiar with the IoT space?

now. And it's poised for explosive growth. But I do think the importance of IoT should be reinforced at events such as JavaOne and Oracle OpenWorld, with keynote addresses and sessions. That would be a great forum for this kind of promotion.

I would compare IoT right now to the initial phase of the smartphone app market. There was some initial resistance, and nobody knew exactly how things would evolve. But apps very quickly became a multi-billion-dollar market. I believe that will happen for IoT as well.

Java Magazine: Could you discuss the interplay between standards and innovation in the world of Java and the JCP?

Patel: Too much rigidity can be the enemy of innovation. But we also need well-crafted standards to attract developers around a common framework. Millions of developers worldwide are affected by the standards developed through the JCP. The platform must function as an integrated totality. And it does. But it must also be flexible enough and nimble enough to respond to dynamic market forces. We talk about that frequently within

PATEL ON THE JVM

“In the world of IoT development, there is sometimes a concern that the JVM will be an impediment to efficiency. But evolving Java standards are making Java ME applications very efficient.”

the JCP community. And I think we're generally doing a good job of achieving those goals.

In the world of IoT development, there is sometimes a concern that the JVM [Java Virtual Machine] will be an impediment to efficiency. But evolving Java standards are making Java ME applications very efficient.

It's also important to consider overall system cost. This is where the Java platform excels. Java not only saves time and promotes reuse, but it also reduces the total cost of ownership. These are huge advantages over a traditional programming scheme.

development saves time and promotes reusability. These are huge advantages in the broader scheme.

Java Magazine: Do you have any closing remarks?

Patel: If we deploy IoT to the lowest common denominator in the human population, that means the household. For instance, if you were driving to work and you had forgotten to set your home alarm, you could pull over and do that on your smartphone. And you could check surveillance cameras to make sure all was well, and then adjust the thermostat or close the garage door. Those myriad devices must be seamlessly connected all the way to the cloud.

There are still connectivity issues to be dealt with in making this happen. A common connectivity framework and architecture will be essential. Multiple connectivity standards are out there—Wi-Fi, Bluetooth, and ZigBee, for instance. Wi-Fi is becoming the most prevalent, but it's not the de facto standard. Connectivity needs to be ubiquitous, highly reliable, and well organized.

And as I've mentioned, security is another concern of primary importance. Customers must be able to trust that their data is secure for IoT applications to really proliferate and become a part of daily life. The evolution of Java through the JCP will be a key element in that success formula. </article>

MORE ON TOPIC:



Steve Melon is a former C/UNIX software developer who has covered the web and the internet for such publications as *Wired*, *Rolling Stone*, *Playboy*, *SF Weekly*, and the *San Francisco Examiner*. He recently published a science-adventure novel, *The Shroud*, and regularly contributes to *The Huffington Post*.

LEARN MORE

- Freescale's Internet of Tomorrow Tour



JAVA IN ACTION



ABOUT US



29

Keba's systems help usher in the next industrial revolution. **BY DAVID BAUM**

ORACLE.COM/JAVAMAGAZINE ////////////////////////////////// NOVEMBER/DECEMBER 2014

KEBA AG

keba.com

Headquarters:

Linz, Austria

Employees:

900

Revenue:

€181 million

Java technologies used:

Java SE 6, Java SE 7, and Java SE 8 with JavaFX, OSGi, and Eclipse e4



the rapidly growing Internet of Things (IoT)—physical objects with network connectivity that can send and receive data. In a modern factory, sensors not only guide the machines, but also provide information to fine-tune the operation as a whole. For example, if the heat or humidity on a production line varies outside of an acceptable range, that line can report this condition and adjust its activities automatically.

The intellectual capital in these versatile factory environments doesn't lie so much with the equipment as with the software that controls it—a pervasive fabric of instructions that binds each robot to the global supply chain. Collecting and analyzing real-time production data dictates many aspects of the manufacturing cycle, from sourcing materials and customizing products to interacting with partners.

Keba AG, an engineering firm based in Linz, Austria, that designs and produces high-tech equipment for the industrial, energy, and banking sectors, is in the vanguard of these modern factory developments. In the field of industrial automation, Keba has built products for controlling injection molding machines and robotic systems. In the energy sector it is well known for its climate control systems and elec-

Keba software engineering team members (from left to right) Hannes Bachmayr, Florian Erler, Martin Fischer, Fabian Schöppl, and Robert Pöchtrager share ideas.



IoT Developer Challenge Winner

Lhings Connected Table

A custom office, wherever you are **BY KEVIN FARNHAM**

The Lhings Connected Table integrates embedded Java, the Raspberry Pi, and the Lhings cloud library with a set of connected devices (RFID cards, lighting, a coffee-maker, and more) to provide a customized office environment that's transferable to any location. Lhings was one of three IoT Developer Challenge winners in the professional category.

"We wanted to explore how IoT [Internet of Things] could be applied to

PHOTOGRAPH BY BOB ADLER/GETTY IMAGES

new areas," explains Lhings CTO José Antonio Lorenzo. "We thought about travelers who work in distinct locations each day and how convenient it would be if their office workspace could be replicated around the world."

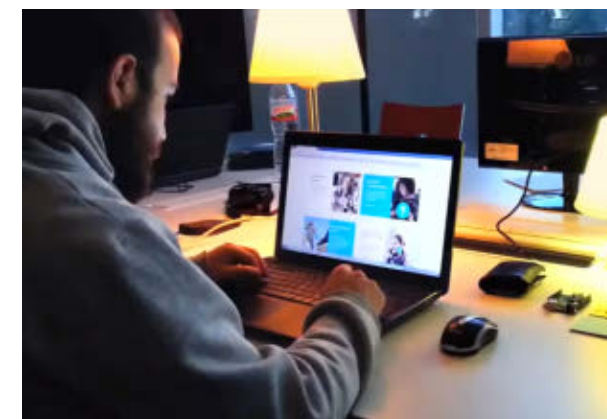
The Lhings Connected Table is a step in that direction. The table lets a user log in to a shared workspace using an RFID card. The user ID information is transferred to the Lhings cloud platform, the user's preferences are retrieved, a text

The Lhings team at JavaOne in San Francisco (from left to right): José Antonio Lorenzo, David Peñuela, and José Pereda

message welcoming the user is sent to his or her mobile phone, and the Raspberry Pi that controls the workspace implements the user's preferences (illumination, documents, printing preferences, and potentially anything else that could be controlled by an IoT device). When the workday is over, the worker scans the RFID card over the tabletop and the workspace returns to its default state, awaiting its next user.

The Lhings platform receives information from internet-connected devices, remotely controls them, and lets users build logic to make the devices work together. "Lhings allowed us to develop the first prototype in only two weeks," says Lorenzo.

"The IoT is totally changing the way people interact with their environment," he adds. "The table is an example of IoT that goes beyond the most-obvious examples. It provides an integral experience in which several devices work together to achieve a common goal, offering the user a customized workspace; delocalizing the desktop; and offering a set of services that make working simpler, convenient, and more productive." **</article>**



Watch the Lhings Connected Table in action.



FAST FACTS

- The Lhings Connected Table lets users customize their workspaces by defining preferences for potentially anything that can be controlled by an IoT device (for example, desktop lighting).
- Preferences are stored in the cloud, and are available around the world at any workspace where the Lhings Connected Table platform is installed.
- *Lhings* is a contraction of *Links things*.



BlueJ brings Java SE 8 development directly to the Raspberry Pi.

Yet the official Raspberry Pi

Then, in the fall of 2013, Oracle released a JDK optimized for the Raspberry Pi ARM platform. Making use of

So all is well, then? Not quite.

When the Raspberry Pi was launched, the founders had a clear vision: They wanted kids to start tinkering with computers and with programming again. The family computer, so the thinking went, is too precious; parents won't let young aspiring programmers play with it for fear that they will muck things up. No



of modularity in a batch of Java Enhancement Proposals (JEPs) that make up the first announced features that are intended to become part of Java SE 9.

This batch includes JEP 201, titled “Modular Source Code.” Despite the name, this JEP does not introduce modularity in any sense that is visible to a developer of Java applications, and the OpenJDK 9 builds as they stand have no trace of modularity in the binaries.

Instead, JEP 201 is a first step that has implemented some necessary housekeeping and has rearranged the layout of the OpenJDK source tree, so that it is now in modules. This JEP has already been completed and is integrated into the JDK 9 source repositories. Let's look at how these new modules are laid out in the OpenJDK 9 source directory (see **Listing 4**).

Inside each module, the directory structure has also been rearranged, with one interesting side effect. The code that was historically common between UNIX-like operating systems was contained in a directory called [solaris](#). With this reorganization, this confusing (and rather legacy) name has been changed to the more accurate [unix](#).

Reinhold also announced a new version of Project Jigsaw, the project aiming to deliver a modular JDK. His vision is of four fairly large JEPs

that will deliver full modularity. The first of these JEPs is, unsurprisingly, JEP 201. The second is JEP 200, which aims to describe and set out the module structure without actually implementing it. The summary of this JEP says,

"Define a modular structure for the JDK. Make minimal assumptions about the module system that will be used to implement that structure."

Within those minimal assumptions is the idea that any module that starts with **java** is one that will be governed by the Java Community Process (JCP) and by Java Specification Requests (JSRs). For developers unfamiliar with the standardization process, what this means is that the **java** modules are those that are intended to be portable across Java implementations (in a future world, in which Java has been fully modularized).

In contrast, the modules we see that start with `jdk` represent internal implementations that are specific to OpenJDK. JEP 200 is still in development, but we can see the first traces of it in the delivery of JEP 201, where the big split between `java` modules and `jdk` modules is obvious.

Conclusion

The arrival of Compact Profiles and `jdeps` represents a great first step

LISTING 4

```
ben$ pwd
/Users/ben/projects/openjdk/jdk9/jdk/src
ben$ ls
bsd                java.rmi           jdk.charsets
jdk.jconsole       jdk.security.auth
demo               java.scripting     jdk.compiler      jdk.jdi
jdk.snmp
java.base          java.security.acl  jdk.crypto.ec
jdk.jdwp.agent     jdk.zipfs
java.corba         java.security.jgss jdk.crypto.ms capi
jdk.jvmstat        linux
java.desktop       java.security.sasl jdk.crypto.pkcs11
jdk.localedata     sample
java.instrument    java.smartcardio   jdk.deploy.osx
jdk.naming.dns     solaris
java.logging       java.sql           jdk.dev
jdk.naming.rmi
java.management    java.sql.rowset    jdk.hprof.agent
jdk.rmic
java.naming         java.xml.crypto    jdk.httpserver
jdk.runtime
java.prefs         jdk.attach        jdk.jcmd
jdk.sctp
```



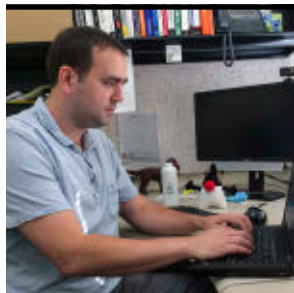
Download all listings in this issue as text

toward a more modular future, but we still have a long way to go before we see a fully modularized JRE. In particular, the build process needs to be updated to produce modular binary artifacts (and the end of a monolithic [rt.jar](#)), and the class-loading subsystem needs to become aware of modules as a primary mechanism for loading code into a Java process. In the coming months, a great deal of activity is to

be expected, with further updates from the platform team as Project Jigsaw progresses and the modularity train starts to pick up steam again. **</article>**

LEARN MORE

- [jdeps](#)
- [JEP 201: Modular Source Code](#)
- [JEP 200: The Modular JDK](#)



ERIK COSTLOW AND
ERIC RENAUD

BIO

ERIC RENAUD PHOTOGRAPH BY
VOLTIRE YAP

JDK 8u20 Improves Performance, Security, and Manageability

Learn how to better control managed systems that run multiple rich internet applications.

Following the Java 8 release earlier this year, Java SE Development Kit 8, Update 20 (JDK 8u20) continues to improve upon the significant advances made in the Java SE platform. As the latest minor release of Oracle's implementation of Java SE, JDK 8u20 was architected, in part, to provide enterprise system administrators with the ability to better control managed systems on which users run multiple rich internet applications (RIAs), specifically Java applets and Java Web Start applications.

In this article, we will explore the two new tools most beneficial

to such administrators, the [Java Advanced Management Console](#) and the [Microsoft Windows Installer Enterprise JRE Installer](#), along with other important improvements in the JDK 8u20 release.

REDUCE RISK
By using the Advanced Management Console, administrators can reduce security concerns by effectively creating whitelists and blacklists.

Advanced Management Console

First, let's address the Advanced Management Console. It is available in the [Oracle Java SE Advanced products](#), which provide enterprises and independent software vendors (ISVs) with support and specialized tools such as the Advanced Management Console and

Oracle Java Mission Control.

The functionality of Advanced Management Console 1.0 can be boiled down to two areas. It directly tracks usage data for Java applications, and it enables administrators to act on that data in a guided manner. Using the Advanced Management Console results in a more easily controlled and more secure environment that provides an improved end user experience, as we will see below.

A primary benefit of the Advanced Management Console is the ability to learn which RIAs (Java applets and Java Web Start applications) are being run in an enterprise as well as which Java runtime environments (JREs) are used. Additional information—such as the location of each application, the

vendor, the permission level, and the number of times the application has been run—is also provided, all of which can be gathered from a large number of clients across an enterprise.

How is this accomplished? It's done by way of the [Usage Tracker](#), a Java feature that enables the Java clients on the desktops within an enterprise to report RIA and JRE usage data. The usage data gathered (including the type of virtual machine start, the date and time of the start, the host name and IP address, the application name, and much more) is stored in a normalized database for performance reasons, which means that a single Usage Tracker can handle a large number of clients.

(As an aside, the Usage Tracker is off by default. It is

enabled by creating the properties file <JRE directory>/lib/management/usagetracker.properties. Simply placing that file on a client informs the system to report information to the Usage Tracker. The information is then sent via the User Datagram Protocol [UDP] to prevent any delay on the client. A bit of serendipity is that the properties file can be added by using the new Microsoft Windows Installer Enterprise JRE Installer tool we'll discuss below.)

The data from the Usage Tracker is collected by the Advanced Management Console's Collector, stored in the Advanced Management Console's database, and displayed in the Advanced Management Console's user interface (UI).

Figure 1 and the following steps describe the process for transmitting (reporting) and storing the usage data and making it available for viewing and analysis (visualization).

1. *Reporting:* Java clients are configured with a properties file, either manually per client or automatically across a single or multiple clients through the new Microsoft Windows Installer Enterprise JRE Installer, which tells the clients the location of the Usage Tracker. This file must

be in place for reporting to occur. Then, when the clients start, they provide information about the applications being launched. (As described earlier, the asynchronous UDP packets do not affect the startup performance of the applications.)

2. *Storage*: The Advanced Management Console's Collector gathers information about applications running in the enterprise from Java Usage Tracker via UDP packets and stores the data in the Advanced Management Console's database, managing the growth that can occur for a large user base.

3. Visualization: The Advanced Management Console displays usage information about which applications were used, how often they were used, and with which JRE they were used.

So what does all this usage data do for administrators? It helps them identify the applications that users need to do their work and the versions of Java required to run those applications. And, with security in mind, administrators can also learn whether users are running applications that are not approved or they are running approved applications but using outdated versions of Java.

Figure 2 shows a view of the Advanced Management Console's UI showing the area where usage data would be displayed about applications running on the desktops across an enterprise.

Now, let's look at how the Advanced Management Console enables administrators to take action on the usage information.

The Advanced Management Console uses the Deployment Rule Set security feature, which was introduced in Java SE Development Kit 7, Update 40 (JDK 7u40). The goal of this feature was to help administrators enforce policies regarding which RIAs are allowed or disallowed and which JRE ver-

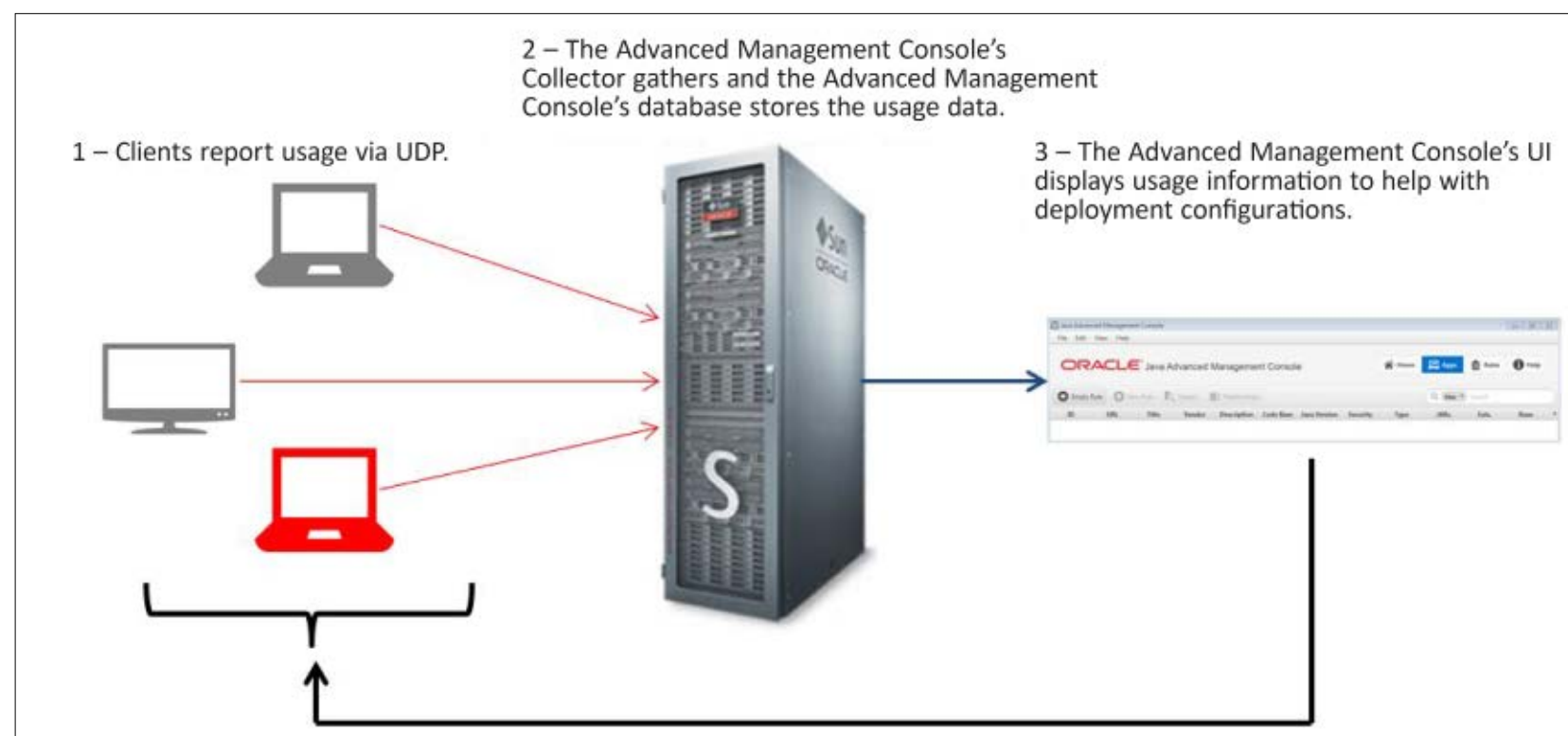


Figure 1

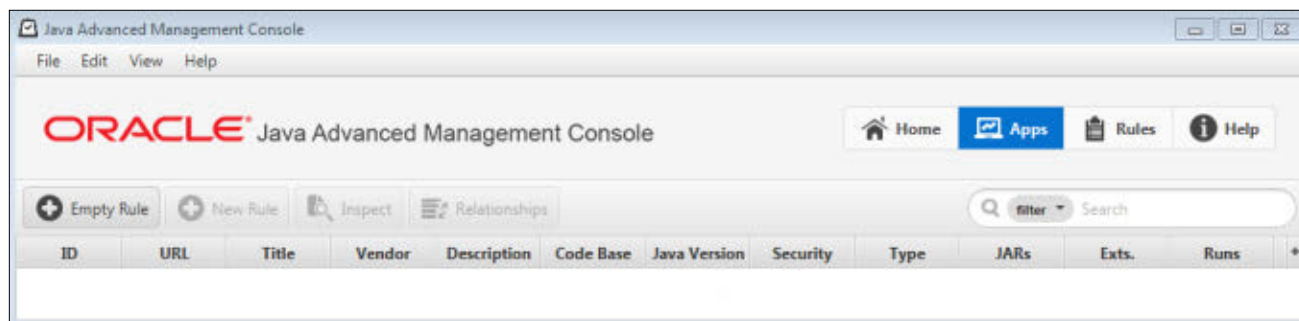


Figure 2

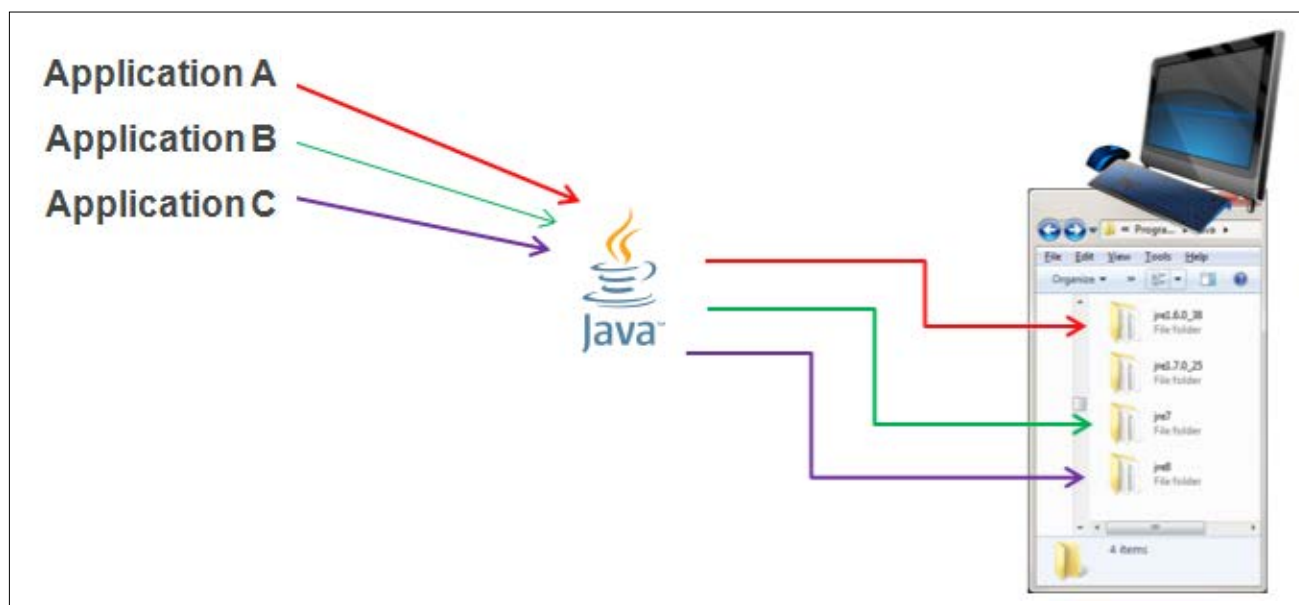


Figure 3

sions can be used to launch them. An additional advantage is that the end user experience is enhanced, because the Deployment Rule Set feature can also control which security prompts users see.

As illustrated in **Figure 3**, the Advanced Management Console's UI provides administrators with the ability to create deployment rules and deployment rule sets and store them in the Advanced Management Console's database, thereby achieving greater con-

control over a more secure network infrastructure.

There are many uses for the Advanced Management Console, but its main advantages are that it equips administrators with a tool set to match applications to desired JREs, to maintain a record of the deployment rule sets that are deployed, and to gather data about applications that are run in the enterprise.

In JDK 8u20, a new “force” feature was added for use with

deployment rule sets, which provides administrators with the ability to specify that an application be run with a different JRE version than the version specified in the application itself.

Deployment rule sets also offer a secure way of managing compatibility with older versions of Java. By using a deployment rule set, the latest and most secure version acts as a proxy to allow only “known to be safe” applications to run with older, compatible JRE versions. As a result, most applications use the current, secure JRE and older JREs are limited to running “known to

be safe” applications. Similarly, a deployment rule can be deployed that causes the “last known to run” JRE to be used for a particular application while keeping all the other applications on up-to-date JREs. Thus, by using the Advanced Management Console, administrators can reduce security concerns by effectively

creating whitelists and blacklists.

An important item to note is how guided rule creation and packaging support greatly simplify developing deployment rule sets. The Advanced Management Console can also be used to determine which rules and rule sets an application matches, helping system administrators understand the impact of installing a particular rule set prior to actually testing the rule set in user environments. During the guided creation of deployment rule sets, Usage Tracker data identifies applications by certificate hash and by location.

In addition, a comparison tool is available to verify rules against tracked data, thus enabling easier testing. While administrators can always create deployment rule sets by hand using a text editor, these new tools greatly reduce the time and effort required and make the process far less error-prone.

In summary, the Advanced Management Console enables system administrators to easily identify RIAs and JREs, and it provides tools for controlling the com-

TRACK AND ACT

The functionality of Advanced Management Console 1.0 can be boiled down to two areas.

It directly tracks usage data for Java applications, and it enables administrators to act on that data in a guided manner.

patibility and availability of older Java installations through deployment rule sets in a scalable manner across the enterprise, all of which results in a streamlined experience for users. **Figure 4** shows a view of some rules and rule sets that have been deployed in the Advanced Management Console.

Microsoft Windows Installer Enterprise IRE Installer

The new Microsoft Windows Installer Enterprise JRE Installer integrates into various desktop management tools, making it easier to customize and roll out different Java SE versions.

Available in the Oracle Java SE
Advanced products for Windows

64- and 32-bit systems, this new installer provides a number of benefits for system administrators who customize or manage software in the enterprise at scale. Unlike the basic installer that most users obtain from Java.com or Oracle Technology Network, this installer is built around customization and integration with various desktop management products such as the Microsoft System Center Configuration Manager (SCCM), and it provides automated, consistent installation of the JRE across all the desktops in an enterprise.

System administrators who use the Microsoft Windows Installer Enterprise JRE Installer can use every capability provided by the

standard [Windows Installer](#), such as silent installations and upgrades, low-privileged installations, and self-repair capabilities. Other common features—such as rolling back unsuccessful installations, repairing broken installations, and installing over existing broken installations—are all available with the Microsoft Windows Installer Enterprise JRE Installer. Integrated with the installer is the Java Uninstall tool, which provides the option to remove older versions of Java from a system.

Other Improvements

JDK 8u20 also provides enhancements that improve performance, such as a reduced memory footprint, several patches geared to next-generation CPUs, and improved support for garbage-first (G1) garbage collection for long-running applications.

For G1 garbage collection, the newly added String deduplication capability described in [JDK Enhancement Proposal \(JEP\) 192](#) does not actually deduplicate the String objects; it deduplicates only their backing character arrays. However, the resulting optimization is elegant, removes inefficiency, and results in heap reduction.

Conclusion

While the enhancements and optimizations in JDK 8u20 are numerous and impactful, it is the new tools it provides in Oracle Java SE Advanced products that add the luster to this release. The new tools make it easier for system administrators to identify and control client installations at scale. Administrators at organizations that want either the tools or associated commercial support should consider using Oracle Java SE Advanced products.

Oracle Java Mission Control (featured in the [July/August 2014](#) issue of *Java Magazine*) continues to be available as a commercial feature in the Oracle Java SE Advanced products. The new version, [Oracle Java Mission Control 5.4](#), is bundled with JDK 8u20 and includes several enhancements to improve usability.

Advanced Management Console and Oracle Java Mission Control require an Oracle Java SE Advanced product license for production use, but each tool is available for download for development and evaluation purposes from Oracle Technology Network. **</article>**

LEARN MORE

- [JDK 8u20 Update Release Notes](#)
- [Oracle Java SE Advanced](#)

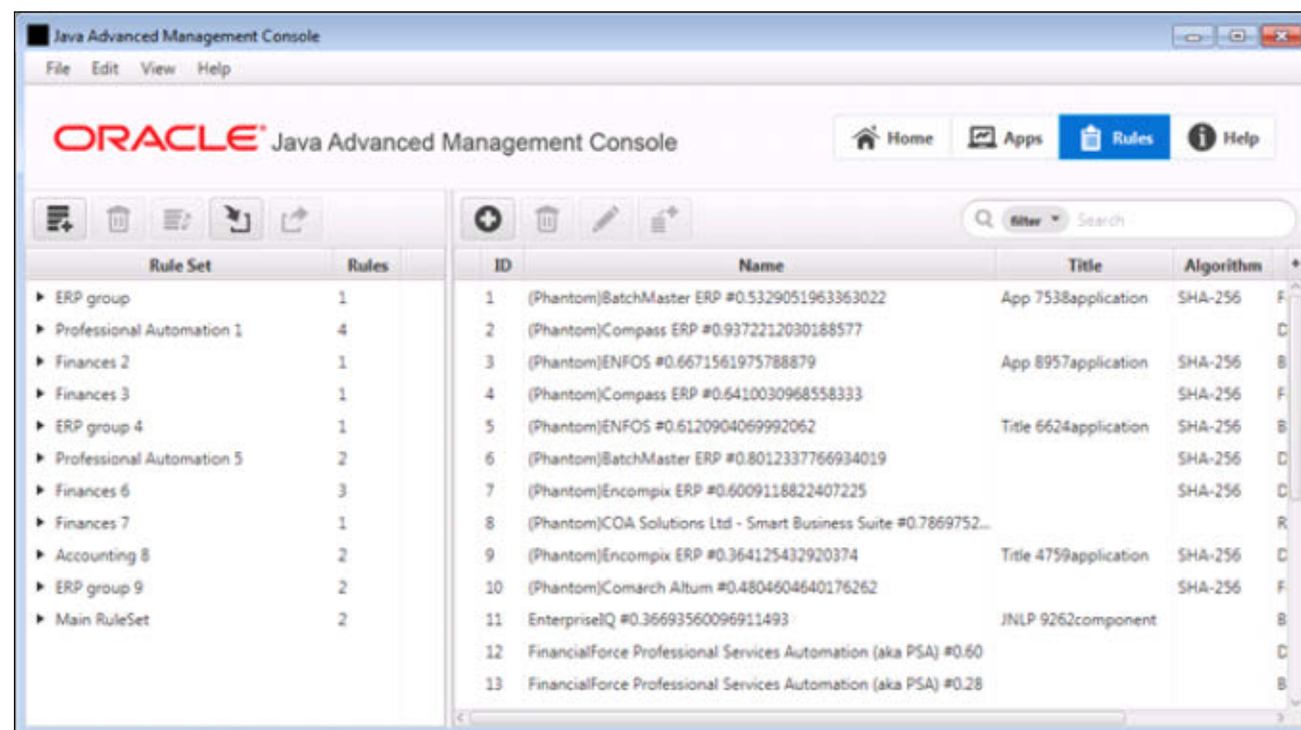


Figure 4



Eclipse SmartHome bridges the gap between tech-savvy users and average users to provide a smart-home platform for everyone.

2014 is the year of the Internet of Things (IoT). According to Gartner, hype about the IoT is currently at a peak, which means the market has huge expectations that soon might give way to some disillusionment.

But let's begin with what is currently available. Smart-home technology has existed for a few decades already, but it has never found its way into the mass market. There are actually two main segments:

- openHAB has attracted a large community of DIY users. The project serves as a central integration point where different home automation systems and protocols can be combined into one overarching solution that provides uniform user interfaces across all devices and, more importantly, a possibility to define automation logic that bridges all gaps. All this is done through textual configuration and scripting, which is flexible and powerful for the



PHOTOGRAPH BY
TON HENDRIKS

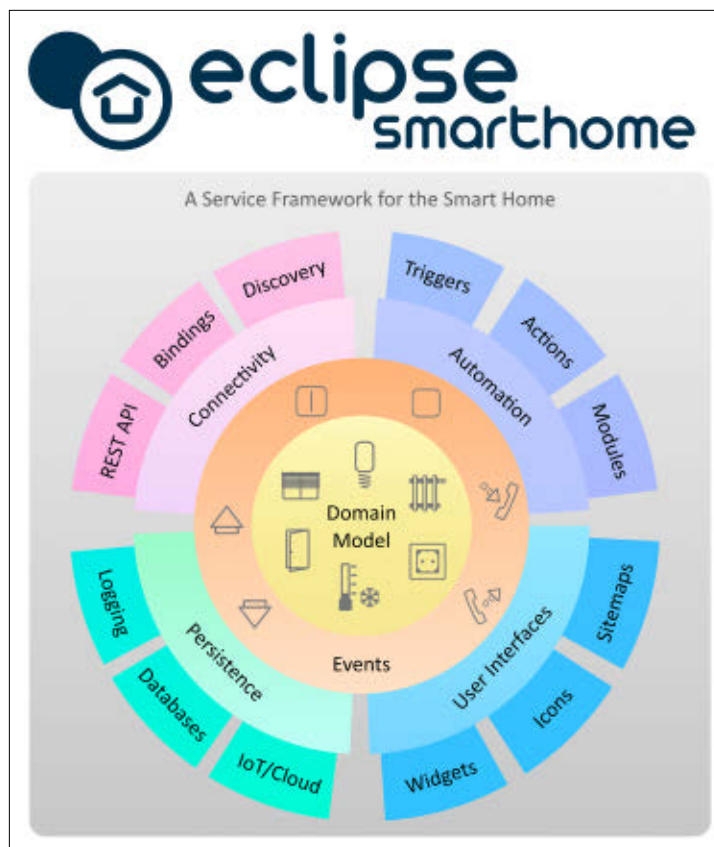


Figure 1

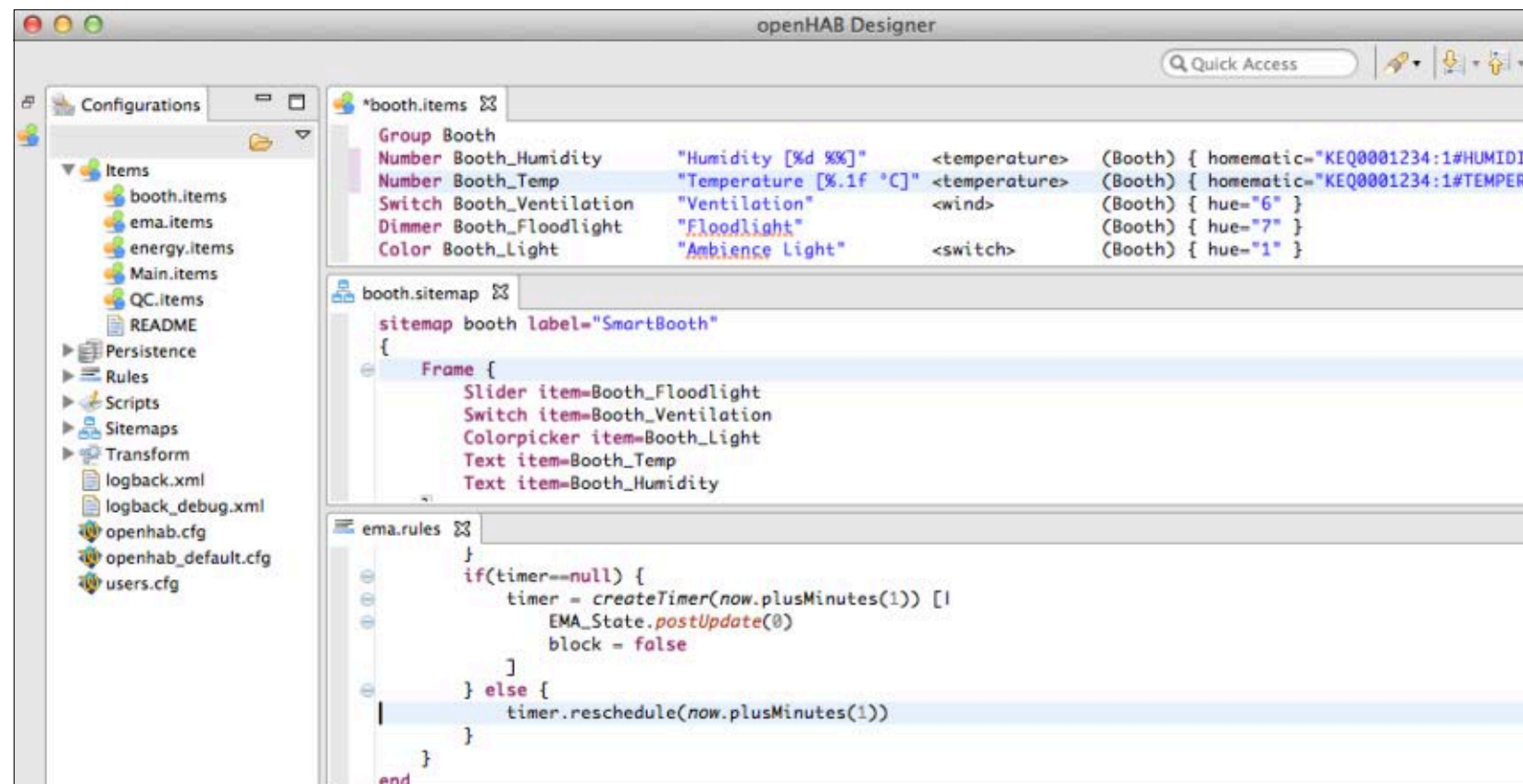


Figure 2

tech-savvy user but daunting for the average user. To address this problem and broaden the potential user base, the code for the open-HAB framework was contributed to the Eclipse Foundation in early 2014 as an initial seed for the Eclipse SmartHome project.

A Home Gateway Stack

The Eclipse SmartHome project is meant to be a software framework tailored for home gateways. It provides APIs and services for four categories: connectivity, automation, user interfaces, and persistence (see **Figure 1**).

Connectivity covers not only communication via a certain protocol or a proprietary interface among devices or systems, it also deals with making the gateway accessible in a homogenous way for client applications. Just as the framework expects other systems to provide open APIs, it itself exposes its functionality through a REST API.

Automation is all about rules and the smart parts of a smart home. In essence, this is the code that brings the whole system to life. Homes are extremely individualistic, and no two setups will ever look alike. Therefore, a

paramount requirement is that the framework offers flexible mechanisms for implementing logic. This requirement can be reached only by using some kind of programming language.

User interfaces (UIs) are the visual component of a smart-home solution. To appeal to users, it is important that UIs offer a consistent way of accessing devices and visualizing the current state of devices. Ideally, there is one UI to address all needs, not a plethora of different apps. As a result, such a UI must be generic enough to easily adapt to new kinds of devices and use cases.

Persistence transforms ephemeral events into permanent information. Persistence can be provided through local databases or cloud services, and the data can be used for reports or charts or for inferring automation actions. Such persistence services must be configurable with respect to which data should be collected (for example, every changed value) and with which kind of strategy (for example, at a fixed frequency).

All these aspects were addressed in a textual way in openHAB, as can be seen in **Figure 2**. For Eclipse SmartHome, completely new APIs



JAVA TECH

ABOUT US



blog



49

MORE ON TOPIC:





I²C sensor using the Device I/O API, your code will be 100 per-cent portable to different target boards such as the Raspberry Pi, BeagleBone Black, UDOO, and dozens of other boards.

This article will present Device I/O API details, code for a practical project, and also some examples of industry applications.

Device I/O API Features

The Device I/O API abstraction is very complete and includes support for analog-to-digital converters (ADCs), digital-to-analog converters (DACs), AT command devices, counters, GPIO pins, I²C devices, pulse-width modulation (PWM) generators, SPI devices, UART devices, watchdog timers,

and modem signal control. Beyond providing low-level control, the Device I/O API also provides support for security, concurrency, and power management, which will bring the next level of device access to Java SE.

Another important thing about this API is that it is open source. Feel free to participate

and collaborate; the official web-site is <https://wiki.openjdk.java.net/display/dio/Main>. There is an amazing team supporting and writing this API.

Getting Started with the Device I/O API and the Raspberry Pi

You can start playing with the Device I/O API and the Raspberry Pi today, because Oracle is writing the implementation for Linux with hard-float support and is testing the implementation with Raspberry Pi boards (based on the BCM2835 system on a chip [SoC]). You will need the following items:

- One Raspberry Pi, Model A, B, or B+
- One secure digital (SD) card with a Raspian image
- An LED or any other component you want to control
- Wires
- A protoboard (optional, but recommended)

Once you have your board up and running, you will need to download and install JDK 8 for ARM.

Additionally, you need to install Mercurial to check out the Device I/O API source code. To install it, just type the following command at your Raspberry Pi terminal:

```
sudo apt-get install mercurial
```

Then, you can get the source

LISTING 1

```
hg clone http://hg.openjdk.java.net/dio/dev
```



Download all listings in this issue as text

code by typing the command shown in **Listing 1**.

You might need to install the GNU Compiler Collection (GCC) or some additional tools, but typing the following commands should be enough to build a Device I/O project for most Raspbian distributions:

```
export PI_TOOLS=/usr
export JAVA_HOME=<your jdk path>
cd dev
make
```

Now you can run some sample applications, check the API details, and start coding. The most important files you will need are the following:

- `/dev/build/so/libdio.so` is the compiled C native code.
- `/dev/build/jar/dio.jar` is the complete Java Device I/O API.
- `/dev/build/jar/dio-samples.jar` is a sample application using an LED and an SPI.
- `/dev/samples/gpio/gpio.policy` is the policy file template.
- `/dev/config/dio.properties-raspberrypi` specifies the default configuration for the

Raspberry Pi.

Inside the `dev` directory, you will find the following subdirectories:

- `/dev/src`, which contains the Java and C native code
- `/dev/test`, which contains the test suite
- `/dev/build`, which contains all the compiled files, the `linux.so` library, and the `.jar` files
- `/dev/build/deviceio`, which contains compiled files with directory structure prepared to be installed in your JRE
- `/dev/config`, which contains the specific property configurations for Raspberry PI devices and GPIO

The `src` subdirectory is organized as follows:

```
|----se
|  |----classes
|  |----native
|  |----linux
|
|----share
|  |----classes
|  |----native
|  |----linux
```

The `share` subdirectory contains

ON THE RISE

The number and types of single-board computers are growing fast, and each type provides a different API for using its GPIO pins.

all the base source code for the Device I/O API that can be shared with other platforms, while the [se](#) subdirectory contains specific classes and native code for the Oracle Java SE Embedded platform. The main package for the Device I/O API is [jdk.dio](#), and you will find all the interface definitions inside [share/classes/jdk/dio](#).

Writing Your First Device I/O Application

To run your first application using the Device I/O API, you will need the following:

- Any standard Java class with a **main** method
 - **java.policy**, which is a file that contains the permissions configuration
 - **dio.jar**, which contains the Java Device I/O API itself
 - **libdio.so**, which contains native code
 - **dio.properties**, which contains board-specific configuration (for the Raspberry Pi)
- Let's write the code for using a blinking LED with an embedded

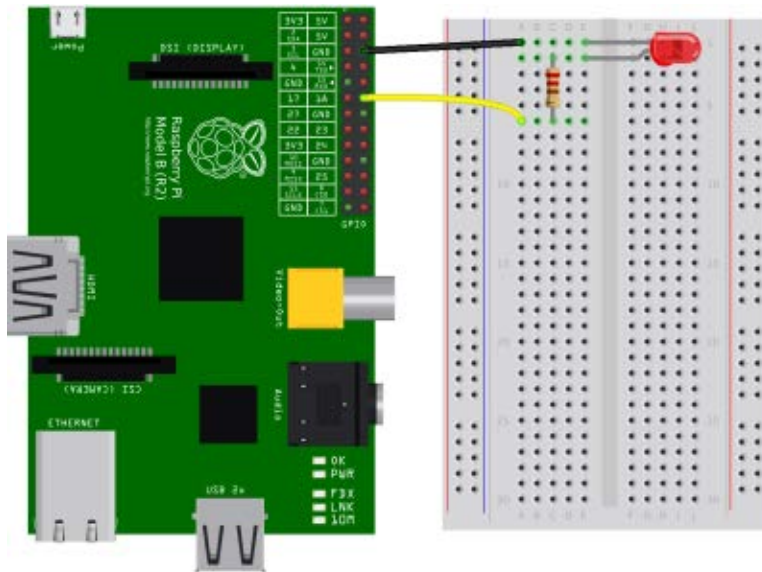


Figure 1

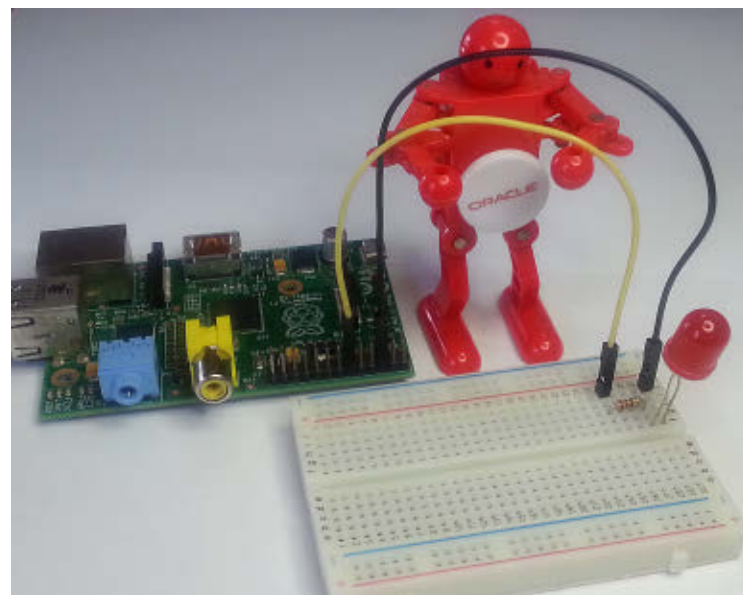


Figure 2

version of the well-known embedded Hello World program.

In terms of hardware, you will need the following:

- A Raspberry Pi, any model
- An SD card
- A power supply (5 volts, 1 amp)
- One LED

LISTING 2

LISTING 3

LISTING 4

LISTING 5

```
package javamagazine.dio;

import jdk.dio.DeviceConfig;
import jdk.dio.DeviceManager;
import jdk.dio.gpio.GPIOPin;
import jdk.dio.gpio.GPIOPinConfig;

public class LedBlink {

    public static void main(String[] args) throws Exception {
        GPIOPin pin = null;
        pin = (GPIOPin) DeviceManager.open(18);
        System.out.println("Blinking LED");
        for (int x = 0; x < 20; x++) {
            pin.setValue(x % 2 == 0);
            Thread.sleep(1000);
        }
        pin.close();
    }
}
```

 [Download all listings in this issue as text](#)

- One resistor (220 ohms)
- A protoboard and male-female wire

You can wire your protoboard as shown in **Figure 1** and **Figure 2**.

As you can see, the code in **Listing 2** is very simple. We have the `DeviceManager` class as a starting point to establish a connection between the Java object named `pin` and the physical device, in this case, GPIO pin number 18.

To run the application, you must run the command with the argu-

ments shown in **Listing 3**.


To make things easy, you can install Device I/O files in your Java runtime environment (JRE), because we have the directory `/dev/build/deviceio` prepared to install `dio.jar` and `libdio.so` in your JRE, and it includes `libdio.so` in `lib/ext/arm` and `dio.jar` in `lib/ext`. To do this, just type the command shown in **Listing 4**. Then, you won't need to specify `dio.jar` and the additional library path (see **Listing 5**).

The security policy allows the Java Virtual Machine (JVM) to

In practice, if you have a Java ME or Java SE project that reads an I²C sensor using the Device I/O API, **your code will be 100 percent portable to different target boards** such as the Raspberry Pi, BeagleBone Black, UDOO, and dozens of other boards.

To finish, let's see what the Raspberry Pi properties file looks like (see **Listing 9**). This file declares and configures all the devices we have on our board, from pins to watchdog timers, UART devices, and so on. The device numbers will vary according to the board or SoC you are using, so you must check the data sheet and manuals if you are going to

To run this application, you can use the command shown in

 [Download all listings in this issue as text](#)

The Device I/O API supports much more than GPIO pins; most of the popular device communication standards are supported. **Listing 11** shows the code for reading a GPS sensor using UART communication through Raspberry Pi pins 8 (RX) and 10 (TX).

We are living in an exciting time for the embedded space as a result of

INTERFACE	DESCRIPTION
jdk.dio.adc.ADCChannel	ANALOG-TO-DIGITAL CONVERTER SUPPORT
jdk.dio.atcmd.ATDevice	MODEM/CELLULAR SUPPORT
jdk.dio.counter.PulseCounter	PULSE COUNTER SUPPORT
jdk.dio.dac.DACChannel	DIGITAL-TO-ANALOG CONVERTER SUPPORT
jdk.dio.gpio.GPIOPin	DIGITAL PIN SUPPORT
jdk.dio.i2cbus.I2CDevice	I ² C BUS SUPPORT
jdk.dio.power.PowerManaged	POWER MANAGEMENT SUPPORT
jdk.dio.pwm.PWMChannel	PULSE-WIDTH MODULATION SUPPORT
jdk.dio.spibus.SPIDevice	SPI SUPPORT
jdk.dio.uart.UART	UART RX AND TX COMMUNICATION SUPPORT
jdk.dio.watchdog.WatchdogTimer	SUPPORT FOR WATCHDOG TIMER THAT FORCES RESTART

Table 1

decreases in hardware prices, the availability of open source hardware, the Internet of Things (IoT), startups, a resurgence of “do it yourself” (DIY) projects, and the Maker Movement. There is definitely an explosion of possibilities that goes much further than the mobile phone market. You can create a gadget to solve a particular problem using sensors, motors, lights, and so on, and doing that is becoming popular. As Arduino’s Massimo Banzi said:

"... hardware becomes like a piece of culture that you share and you build upon, like it was a song or a poem."

The hardware industry now has many small, midsize, and startup companies; it is not like in the past when you could count on your fingers the number of hardware companies. It's predictable that many successful products will comprise custom hardware, code from the community, and the cloud.

Java EE is very reliable and popular for server-side development, big data, the web, cloud, and service-oriented architecture (SOA), and there are now many standards in use. Now it's time to work with standards and implementations for embedded devices. The Device I/O API is definitely an important step

for the future of embedded Java, similar to what JDBC was for the Java platform.

Conclusion

This is just the beginning for embedded Java. Much more will come from the community, companies, and universities. If we look again to the Java EE platform, after JDBC many persistence frameworks were created and now we have a standard for persistence frameworks: Java Persistence API. The same might happen in the embedded space; once the standard API for Device I/O control is done, we might see more and more frameworks for devices in the embedded space, making it even easier to work with hardware, boards, devices, and embedded Java.

Now—what will you create with Oracle Java SE Embedded and the Device I/O API? **</article>**

MORE ON TOPIC:



LEARN MORE

- [OpenJDK page for the Device I/O project](#)
- [Video: Accessing HW Devices Using Java ME 8 Device I/O API](#)
- [Download the code shown in the video demo](#)

A man in a blue hoodie is shown in profile, looking out over a city skyline. Overlaid on the image is a network of orange hexagons connected by lines. Some hexagons contain icons: a building, a coin, an ATM, and a laptop. The background is a light blue sky with a city skyline, including the Transamerica Pyramid.



Use JavaFX 3D to model historical treasures and more.

When geomatic engineering is applied to the reconstruction of cities, historic buildings, or museums, digital documentation—such as an inventory of the buildings and sites—is generated. Then, with the use of visualization tools, georeferenced and scaled virtual tours can be created, allowing you to “walk” through the areas while sitting in front of your computer.

GET A VISUAL
JavaFX 3D can be used as the **visualization framework** for displaying a 3-D model and navigating through it.

reverse engineering and a hybridization of technologies and techniques such as geographic information systems (GIS), artificial vision, terrestrial photogrammetry, and flight with microdrones. Then they postprocessed the data for point cloud optimization, orthorectified the images, and

This article focuses on the visualization tool, [JavaFX 3D](#), which was used as the visualization framework for displaying the 3-D model and navigating through it. By using the JavaFX 3D API, you can develop your own applications for targeting and customizing some of the usual capabilities of commercial software.

Neither digital images nor analog images are metric. In other words, we cannot derive measurements of the objects that are displayed nor can we derive the distances between them. We can visualize, interpret, and analyze, but we cannot exploit the objects metrically. Computer stereo vision is the extraction of 3-D information from digital images. Through multiple images of a scene taken from different viewpoints, the 3-D

Photogrammetry is a technique for obtaining two- or three-dimensional metric information from photographic images. Photogrammetry is, therefore, a technology that can provide dimensional aspects when using images to reconstruct a given scene. By using oblique processing techniques for 3-D reconstruction, an object is reconstructed by identifying (in at least two images, and preferably three) the points and lines that form the physiognomy of the object.

Usually, a 3-D modeling project starts with the topographic work, in which total stations are employed to set several main control points and form a network. To create the graphic and photographic documentation of an object, an essential step is the calibration of the camera. For

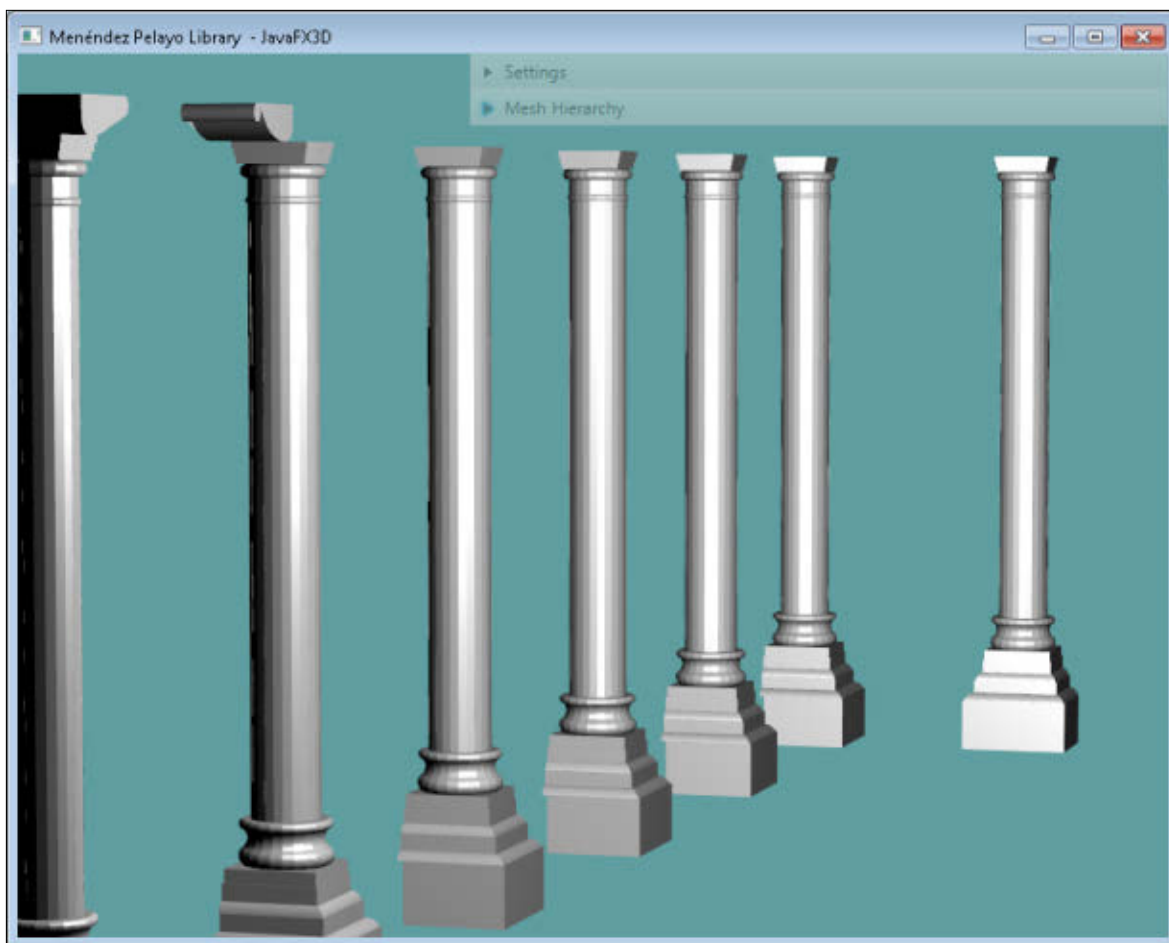


Figure 4

faces consisting of more than four sides. This is the case in **Figure 4**, which shows several columns of the library, which are imported with **PolygonMeshView** after the file **lib_Columns.obj** is imported.

Step 2. Manipulating the models.

When adding different models to a multimodel application, some of the models might require scaling, translating, or rotating to locate them in the correct position. For this, an [Affine](#) class is provided for every model, as shown in **Listing 3**, in which the chair model

is scaled down and translated to its final location.

Using the **prepend** methods causes the transformations to take place in the provided order because, from a mathematical point of view, operations are performed on the left of the transformation matrix (this is called *pre-multiplication*) and referred always to the global coordinate system. In **Listing 3**, the chair is first translated using its actual dimensions (because the original height of the chair's model is 8.43 m and its


LISTING 3

LISTING 4

```
public class Library {
...
    private final ObservableList<Model3D> models=
        FXCollections.observableArrayList();

    public Library(){
        Model3D modelBuildingOut =
            new Model3D("lib_Walls.obj","Library Walls");
        modelBuildingOut.importObj();
        models.add(modelBuildingOut);

        Model3D modelChair1=new Model3D("chair.obj","Chair - 1");
        Affine affineChair=new Affine();
        affineChair.prependTranslation(2,4.13,-12);
        affineChair.prependScale(0.14,0.14,0.14);
        modelChair1.setAffine(affineChair);
        modelChair1.importObj();
        modelChair1.setPickable(true);
        models.add(modelChair1);
        ...
    }
}
```

 [Download all listings in this issue as text](#)

minimum Y coordinate is -4.13 m, it's moved up 4.13 m), and then it is scaled down with a 0.14 factor to a final height of 1.20 m.

Note that by using `append` methods instead, operations would be performed on the right of the matrix, the so-called *postmultiplication*.

tion. This means that new transformations are performed relative to the current object's coordinate origin. While this is the preferred technique in JavaFX, both multiplication techniques are implemented in the JavaFX 3D APIs, so you can use what is best suited for your problem.

the user's pick result, such as the intersected node, the intersected point in the local coordinate system of the picked node, or the distance from the camera to the intersection. **Listing 5** shows how the pick result can be used to move models around the scene. Note that models are moved freely without checking possible intersections with other models.

In the video below, you can see a demo of the application, including several implemented features that are worth mentioning:

- Picking can be used to identify the node and select its tree item in the **TreeTableView**.
- Selecting a mesh tree item on the **TreeTableView** highlights the real mesh.
- In the settings titled pane, mod-

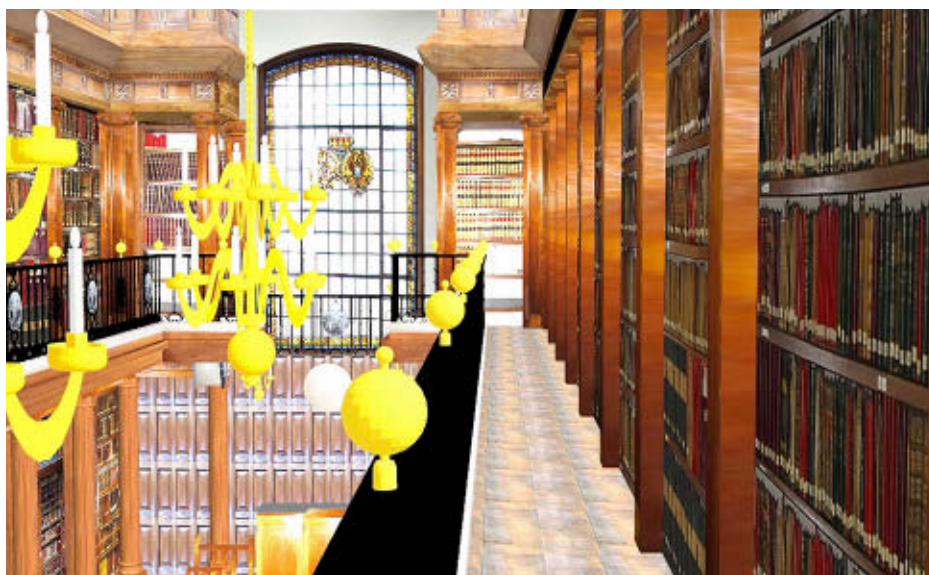
els can be marked as pickable or not pickable, so they can be moved around the scene or prevented from being moved.

- While moving the camera, a balance is required between providing detailed visualization and ensuring usability, especially when dealing with big models. For that reason, textures are unloaded while rotating, panning, or zooming the scene, as shown in **Listing 6**.

Missing features in the application, for example, light or camera manipulation, can be added easily. At this point, you are invited to contribute creating [pull requests](#) with more-advanced features.

Loading the Real Model

Finally, we'll use the developed



José Pereda gives an overview of his article and demos the JavaFX 3D tool.



Figure 6

application to load and visualize the complex model of the Menéndez Pelayo Library created by Óscar Cosido and his team.

Figure 6 shows the outside view of the building, with 10,556 meshes and 1.5 million triangles, which results in a 132 MB `.obj` file and 32 MB in texture images. **Figure 7** shows the interior of the library, with 2,841 meshes and 2.4 million triangles,

resulting in a 143 MB **.obj** file and 90 MB in texture images.

HELP IS HERE

New capabilities
in Java SE 8
(mainly lambdas
and streams)
**help to deal
with** big hier-
archical models.

Conclusion

JavaFX 3D technology is mature enough to enable the development of tools for the geometric processing of complex models obtained by reverse engineering. It provides a way to navigate a virtual model and establishes the basis for the future development of tools for



- ✓ **New Java SE 8 training and certification**
- ✓ **Available online or in the classroom**
- ✓ **Taught by Oracle experts**
- ✓ **100% student satisfaction program**



Learn More

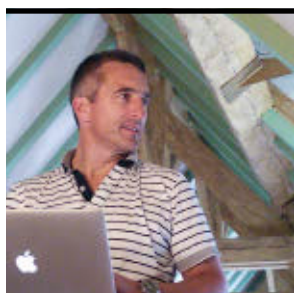


managing 3-D objects in a virtual world as part of a 3-D information system.

thank Óscar Cosido and the students who attended the "GIS and 3D Heritage Modeling" workshop organized by the City Council of Santander. Also, special thanks goes to Esteban Sainz Vidal, director of the CEFEM, for the support and help he offered.

- [OpenJFX project](#)
- [Menéndez Pelayo Library](#)

The author would like to



DAVID GILBERT



A Bridge from Java 2D to JavaFX

Profit from the easy migration path provided by FXGraphics2D.

The JavaFX Canvas is an image node that can be added to a scene to display high-quality text and graphics. It has an “immediate mode” drawing API that resembles the HTML5 Canvas API, and it provides a feature set that is comparable to that provided by the [Java 2D API](#), the standard vector graphics API in Java SE since JDK 1.2. The Canvas node provides a powerful additional capability for developers creating JavaFX user interfaces.

As developers migrate their applications from Swing to JavaFX, though, they might encounter a problem. The JavaFX Canvas API is different than the Java 2D API and does not provide a compatibility layer. Consequently, existing code based on the Java 2D API can require significant rework.

My company, Object Refinery Limited, faced this exact problem with our

library, [JFreeChart](#), which uses the Java 2D API exclusively for rendering. To solve this problem we created the FXGraphics2D project, an open source "bridge" from the Java 2D API to JavaFX. It provides an easy migration path for applications that are

based on the Java 2D API, and
it avoids the need to rewrite
already working code.

This article provides an overview of FXGraphics2D, including a demo and some performance benchmarking. **Figure 1** shows the JFreeChart library being used to render a

chart on a JavaFX Canvas via FXGraphics2D.

About the FXGraphics2D Project

The FXGraphics2D project is small and open source. The library consists of a single class ([FXGraphics2D.java](#))

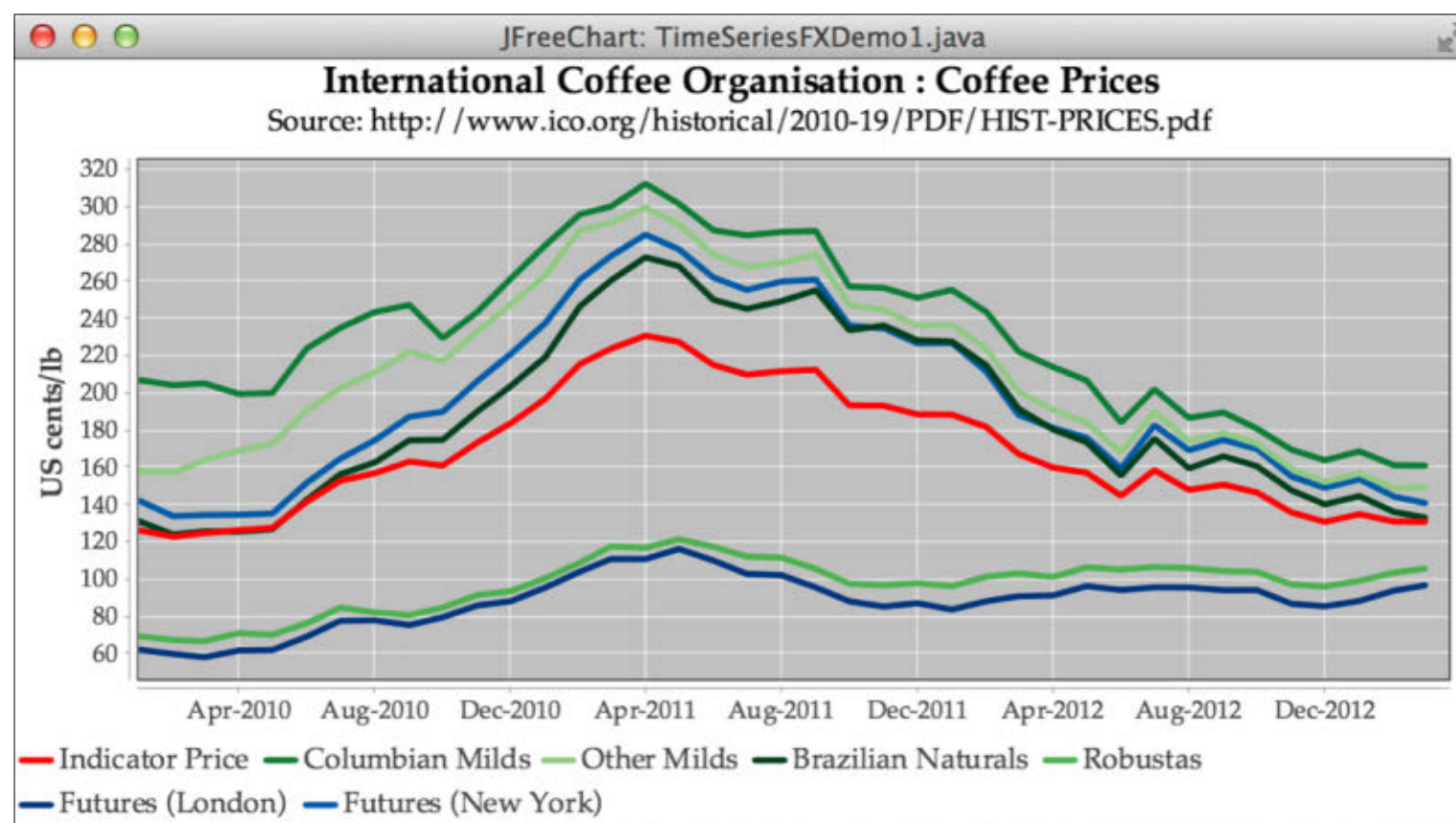


Figure 1

that extends the Java 2D API's **Graphics2D** class and is licensed using a very liberal BSD-style license. We chose this license to ensure that there are no obstacles for anyone who wants to use this code. The source code is hosted at [GitHub](#). The download includes additional files, including some demo applications.

About the Java 2D API

The Java 2D API is the vector graphics API that has been part of the Java standard library since JDK 1.2 was released in 1999. Among other things, it is the graphics engine that powers Swing. The Java 2D API is a transparent, feature-complete, and fast API that is available to all Java application developers, on both the client side and the server side, and on all Java-supported platforms.

Graphics that are compliant with the Java 2D API can be easily exported directly as images in Portable Network Graphics (PNG), Joint Photographic Experts Group (JPEG), and other formats using the

Java Image I/O API. With appropriate third-party libraries—for example, Batik, iText, and JFreeSVG—it is also possible to export to Scalable Vector Graphics (SVG), Portable Document Format (PDF), and other formats easily.

During the last decade and a half, a lot of code has been written to use the Java 2D API—open source examples include JFreeChart, JGraphX, and JLaTeXMath, and there are many more.

Retained Mode Versus Immediate Mode

In JavaFX, the graphics model is scene-based, which means objects are added to a scene and rendering is managed by the JavaFX runtime. Objects can be updated at any time and, because JavaFX retains state information, the scene can be

automatically rendered again without programmer intervention.

JavaFX is sometimes referred to as being a *retained mode* API. Java 2D, on the other hand, is an *immediate mode* graphics API; it does not retain any state infor-

mation about shapes or their properties (color, line style, and so on) after they have been rendered. Immediate mode rendering requires less memory space when rendering complex graphics that have many elements. The JavaFX Canvas node introduces an immediate mode rendering option for JavaFX applications that require it.

A Demonstration

To demonstrate **FXGraphics2D**, we will use the open source JLaTeXMath library, whose stated main purpose is “to display mathematical formulas written in LaTeX.” JLaTeXMath renders its output via the Java 2D API, making it a perfect candidate for use with **FXGraphics2D**. Our simple demo application will produce the output shown in **Figure 2**.

You can download JLaTeXMath [here](#). For this demo, we are using [jlatex-1.0.3.jar](#). In **Listing 1** you can see the code for our JavaFX demo application. There are several important things to note in the code.

In the `start()` method, the first thing we do is preload the fonts that are required by JLaTeXMath. These fonts are included in the `jlatex-1.0.3.jar` file, but they are not available to JavaFX until we load them using `Font.loadFont()`.

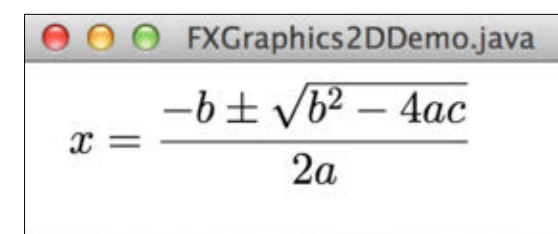


Figure 2

We create an instance of `TeXFormula` using the following string to define the formula that we want `LaTeXMath` to render:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

If you are familiar with LaTeX, you will understand what this somewhat-cryptic string of characters specifies. If not, but you are curious about LaTeX, here is a good starting point.

Next, we create an instance of `FormulaCanvas` (a subclass of the JavaFX `Canvas` class; see **Listing 2**) that handles the creation of an `FXGraphics2D` instance and manages the rendering of a `TeXFormula`.

For this rendering, we create a `TeXIcon` from the `TeXFormula` that was passed to the constructor. This `icon` object can be used easily in Swing because it implements the `Icon` interface, but for our (JavaFX) purposes, a more interesting item is the `Box` object that is returned by the `TeXIcon` `getBox()` method. `Box` is an abstract class in `LaTeXMath` that declares the fol-

KEEP YOUR CODE

Migrating to JavaFX can be a lot easier if you don't need to throw away a lot of existing code. This article demonstrates that **FXGraphics2D is a robust solution** that allows you to keep your tried-and-tested Java 2D code and incorporate it into JavaFX applications.

Create an empty directory named `build` alongside the `lib` and `src` directories, and then compile the demo using the following command:

```
javac -sourcepath src -cp
lib/jlatexmath-1.0.3.jar:
lib/fxgraphics2d-1.1.jar
-d build
src/demo/FXGraphics2DDemo.java
```

... and run the demo with

```
java -cp build:
lib/jlatexmath-1.0.3.jar
demo.FXGraphics2DDemo
```

Performance

A key question that you might already be asking is “how does **FXGraphics2D** perform?” There are two aspects to consider. First, there is the overhead of the on-the-fly translation between Java 2D API calls and the equivalent JavaFX Canvas API calls—surely there will be some cost here. Second, there is the impact of using the JavaFX rendering pipeline versus the Java 2D pipeline. Could this bring some gains?

It would take some careful benchmarking to separately measure these two aspects and, for now, I have not done that. The benchmark used here compares the performance of rendering

directly to a Swing `JPanel` versus rendering to a JavaFX `Canvas` via `FXGraphics2D`—and so both the translation overhead and the graphics engine change are combined in one measure.

We would expect the translation overhead to be minimal in terms of CPU time—particularly as a percentage of overall rendering time—and small in terms of memory space. The impact on memory usage is likely to be the greater of the two overheads, since the calling code will create Java 2D objects (shapes, paths, colors, strokes, fonts, and so on) that will, in many cases, be copied to equivalent JavaFX objects. For most applications, this should not be a big issue.

The switch from the Java 2D rendering pipeline to the JavaFX rendering pipeline is more interesting. I found surprisingly little information online regarding the relative performance of these graphics engines. You might expect the newer JavaFX code to be faster, but bear in mind that the Java 2D API has been the focus of a good deal of optimization over the years. Moreover, JavaFX is required to perform one or two tricks (for example, shadow effects) that the Java 2D API is not required to perform. This can restrict the available options for optimization in certain cases.

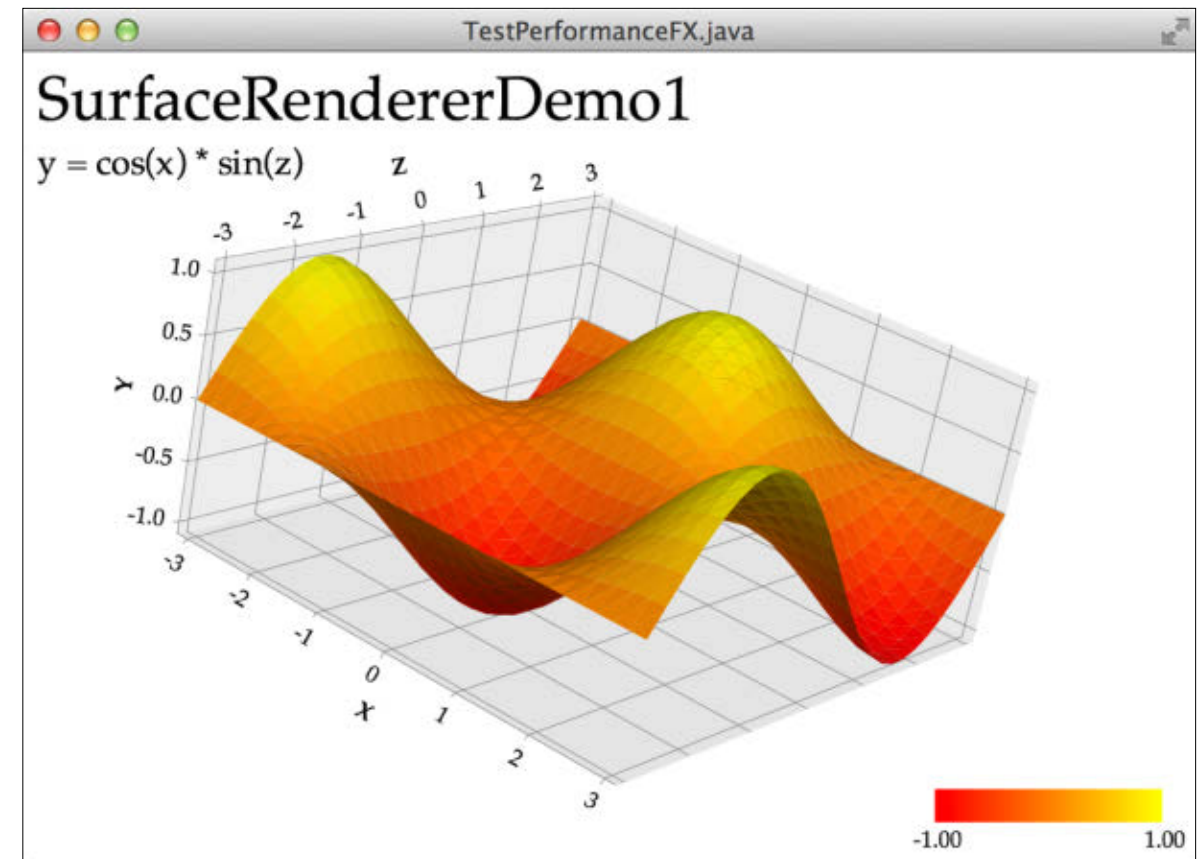


Figure 3

In fact, the one benchmark that I found online suggests that the two graphics engines have roughly equivalent performance, which is not surprising.

We wrote two benchmark programs, one for JavaFX (`TestPerformanceFX.java`) and an equivalent program for Swing (`TestPerformanceSwing.java`). You can download the source code for these programs from [GitHub](#). The benchmark programs render 3-D charts using Orson Charts, a commercial library that is similar in style to JFreeChart (it has the same author: me) but is for 3-D charts

rather than for 2-D charts.

To ensure that anyone can run the benchmarks locally, the benchmarks require only the free evaluation [.jar](#) file for Orson Charts, which you can download.

The benchmark programs create a frame, display a 3-D chart, and then rotate the chart for 1,000 frames. The first 500 frames are considered the warm-up phase to allow the just-in-time (JIT) compiler to work some magic, and the second 500 frames are timed.

Figure 3 shows a screenshot from the JavaFX version (itself a nice example of what can be done

RUN	JAVAFX WITH FXGRAPHICS2D	SWING WITH JAVA 2D
1	18,491 MILLISECONDS	17,623 MILLISECONDS
2	18,636 MILLISECONDS	17,342 MILLISECONDS
3	18,736 MILLISECONDS	17,429 MILLISECONDS
AVERAGE	18,621 MILLISECONDS	17,465 MILLISECONDS

Table 1

with `FXGraphics2D`).
I ran this benchmark on a recent (mid-2014) Apple MacBook Pro with a high-density display on JRE 1.8.0_11.
Table 1 shows the results. Note that you must run the JavaFX benchmark with the following Java Virtual Machine (JVM) option:

```
-Djavafx.animation.fullspeed=true
```

Without that option, you'll get some impressive but misleading results, because the JavaFX rendering engine will populate the rendering queue more quickly than the capped frame rate, and some frames will be dropped, thanks to the `clearRect()` call described previously. For nonbenchmark use, this might be what you want.
As you can see in **Table 1**, the `FXGraphics2D` rendering is—for this particular configuration, at least—slower than the same code in Java 2D, but not by a large mar-

gin. Unless your application has very strict performance requirements, this margin is not likely to be an issue. You should, of course, run your own performance tests with a workload that is representative of your application and target platforms.

Conclusion

Migrating to JavaFX can be a lot easier if you don't need to throw away a lot of existing code. This article demonstrates that `FXGraphics2D` is a robust solution that allows you to keep your tried-and-tested Java 2D code and incorporate it into JavaFX applications.
The `FXGraphics2D` code is free to download, and the license is business friendly. There are no obstacles to using this solution for your own JavaFX applications. `</article>`

- LEARN MORE
- [FXGraphics2D project](#)
 - [JFreeChart](#)

CREATE THE FUTURE

oracle.com/java





In the September/October 2014 issue, Cyril Lapinte gave us code that was not making effective use of streams.

The correct answer is #3: Use the `noneMatch()` method instead of counting the number of filtered results from the inner stream. This will terminate the inner stream as soon as a match is found, meaning that the number is not prime. This will reduce the amount of work done in the inner stream pipeline considerably.

This issue's challenge comes from Abhishek Gupta, a senior identity and access engineer, who gives us a JAX-RS problem.

1 THE PROBLEM

The JAX-RS API (JSR 311) allows us to create resource implementations by inheriting from parent classes decorated with JAX-RS metadata annotations. This lets us enjoy all the benefits of inheritance and isolate the usage of JAX-RS annotations within a single parent class.

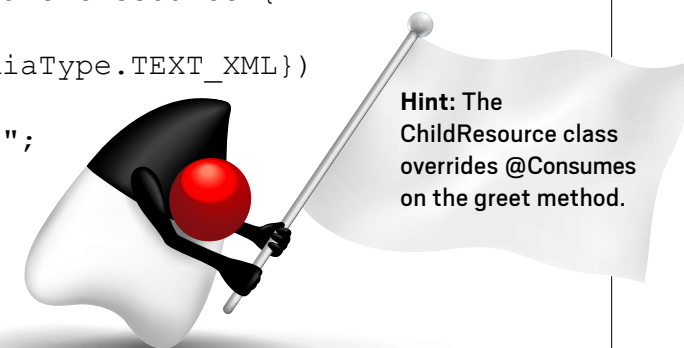
2 THE CODE

The `ParentResource` class is a simple POJO decorated with JAX-RS annotations. Because it is abstract, it has to be extended by a subclass in order to be utilized. `ChildResource` is the subclass that inherits from the `ParentResource` and provides an implementation for its one and only abstract method: `greet`.

Assuming that the REST service is accessible on `http://localhost:8080/TestREST/demo`, what do you think will happen when an HTTP POST request is sent to the service with a text (e.g. Duke) as the HTTP message body payload?

```
public abstract class ParentResource {
    @POST
    @Consumes(MediaType.TEXT_PLAIN)
    @Produces(MediaType.TEXT_PLAIN)
    public abstract String greet(String name);
}

@Path("/demo")
public class ChildResource extends ParentResource {
    @Override
    @Consumes({MediaType.TEXT_PLAIN, MediaType.TEXT_XML})
    public String greet(String name) {
        return "Hello " + name + " !";
    }
}
```



3 WHAT'S THE FIX?

- 1) The invocation results in an HTTP 500 error and is fixed by putting the `@Path("/demo")` annotation on the `ParentResource` instead of the `ChildResource` class.
- 2) The client gets back "Hello Duke!" as the message along with an HTTP 200 response code.
- 3) The invocation results in an HTTP 404 error and is fixed by including the `@POST` annotation on the overridden `greet` method of the `ChildResource` class.
- 4) The application fails to deploy.

Hint: The ChildResource class overrides @Consumes on the greet method.

GOT THE ANSWER?

Look for the answer in the next issue. Or submit your own code challenge!